Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

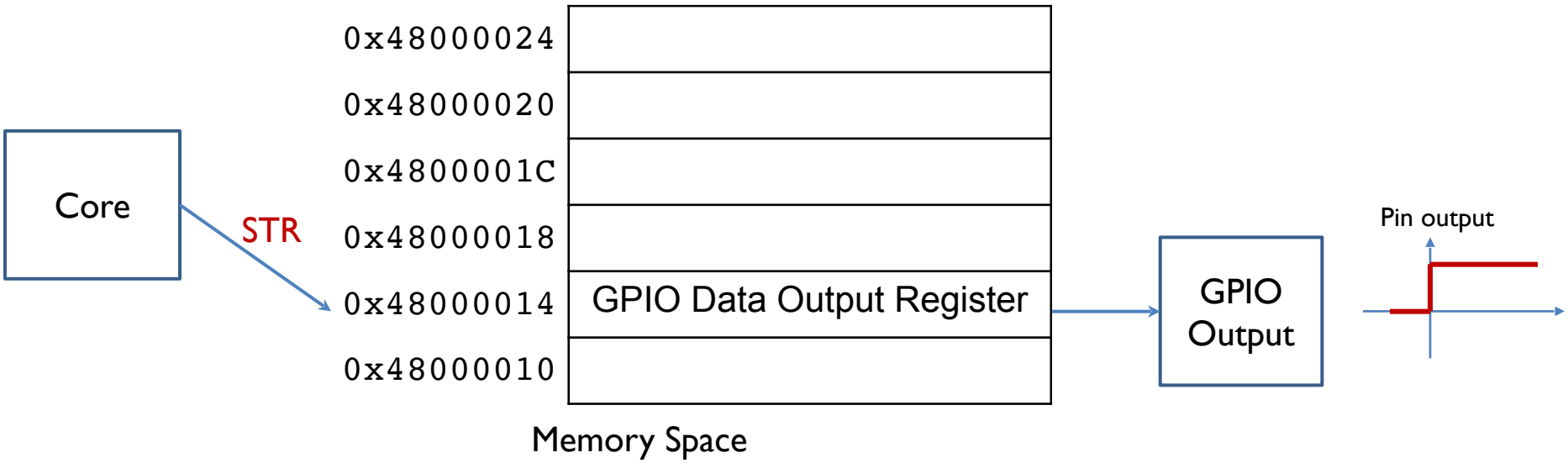
Lecture#16: GPIO Part#1 (Chapter#14)

Spring, 2025

Courtesy: Mohammad A. Qayum, Ph.D.
ECE, NSU, Dhaka, Bangladesh

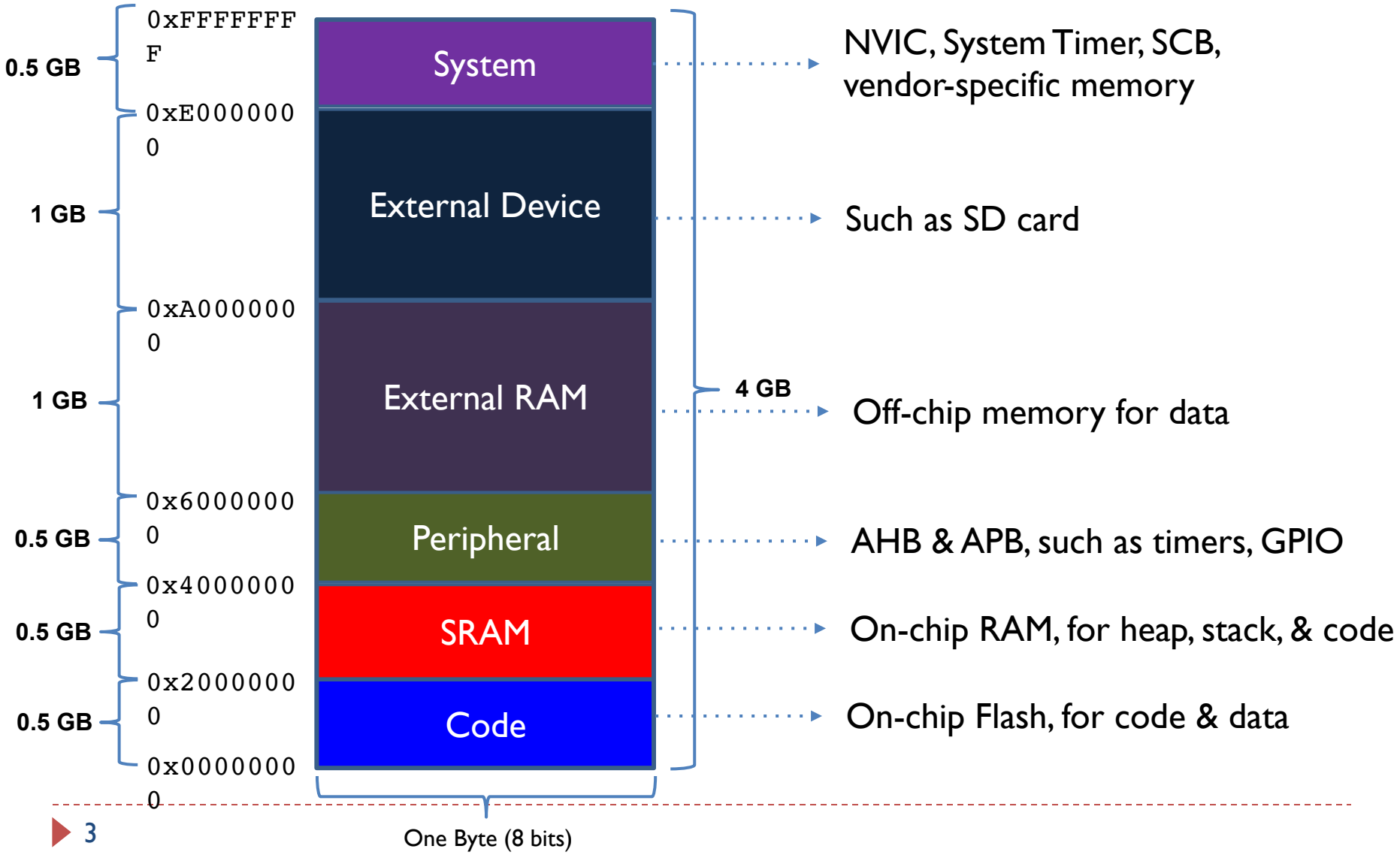
Interfacing Peripherals

- **Port-mapped I/O**
  - Use special CPU instructions: `e.g. Port`
- **Memory-mapped I/O**
  - A simpler and more convenient way to interface I/O devices
  - Each device registers is assigned to a memory address in the address space of the microprocessor
  - Use native CPU load/store instructions: `LDR/STR Reg, [Reg, #imm]`



ARM Cortex-M microprocessors use memory-mapped I/O.

Memory Map of Cortex-M4



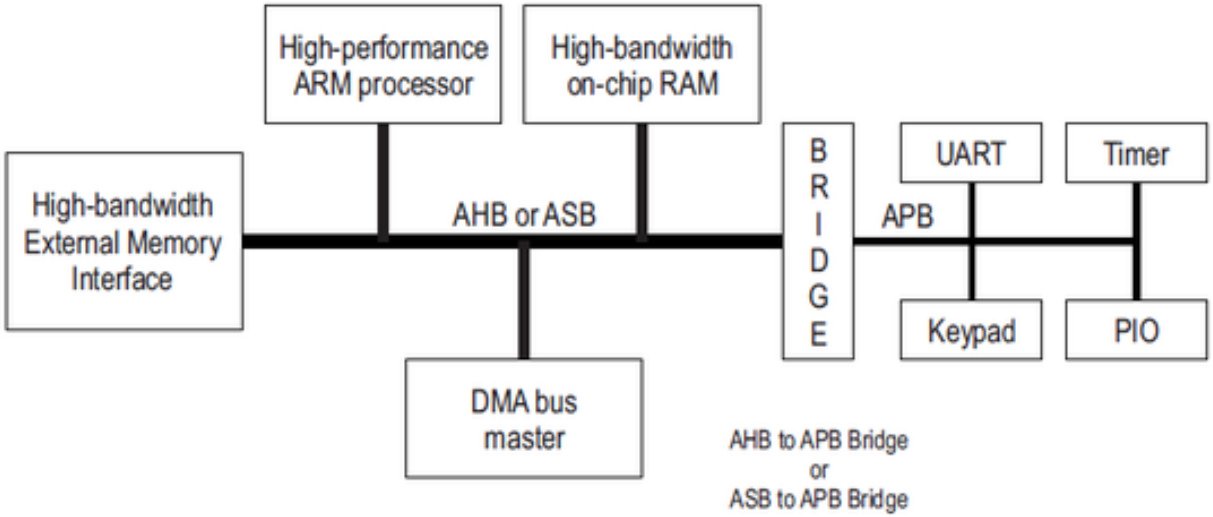
Advanced Microcontroller Bus Architecture (AMBA)

APB: Advanced Peripheral Bus

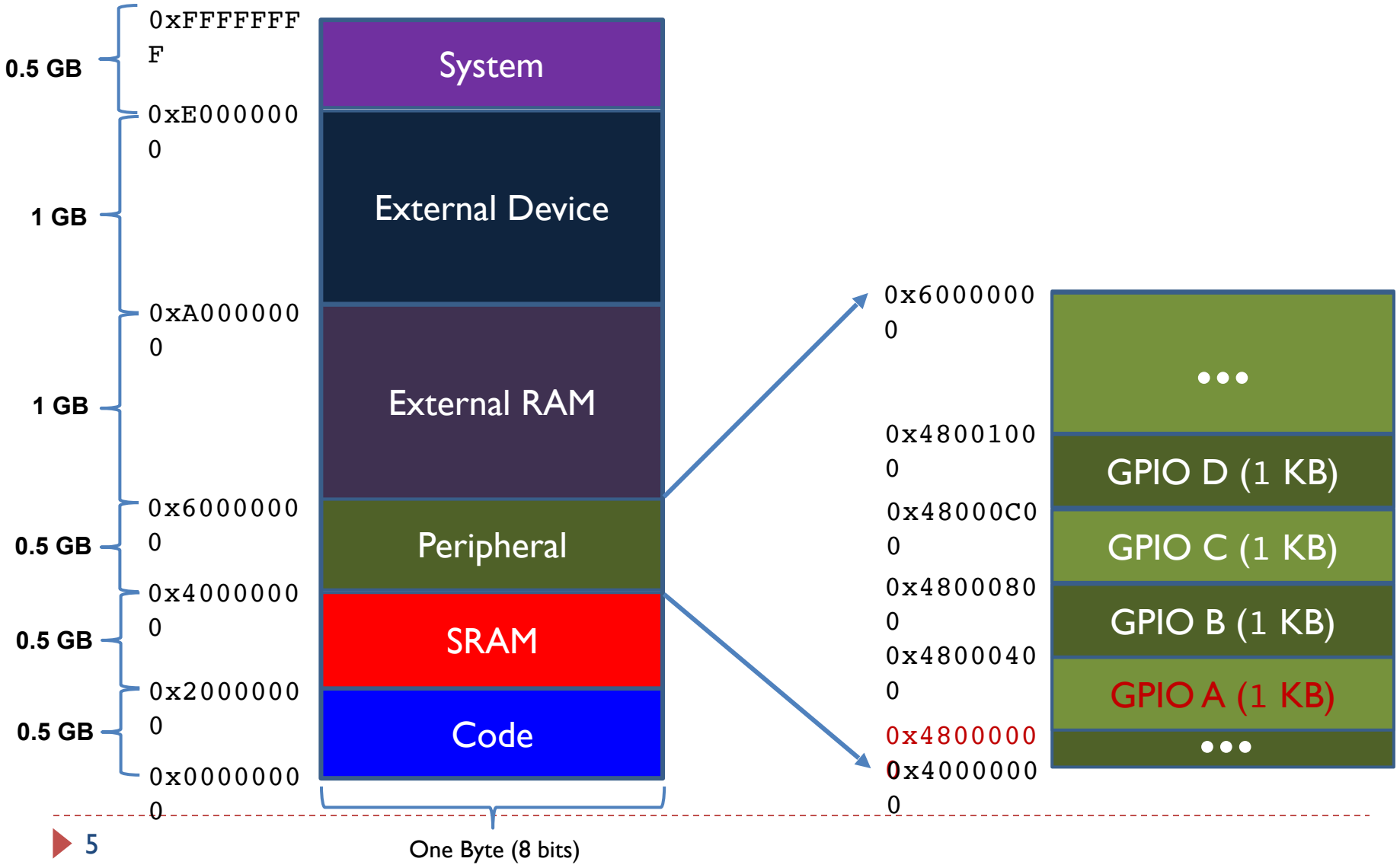
AHB: Advanced High-performance Bus

ASB: Advanced System Bus

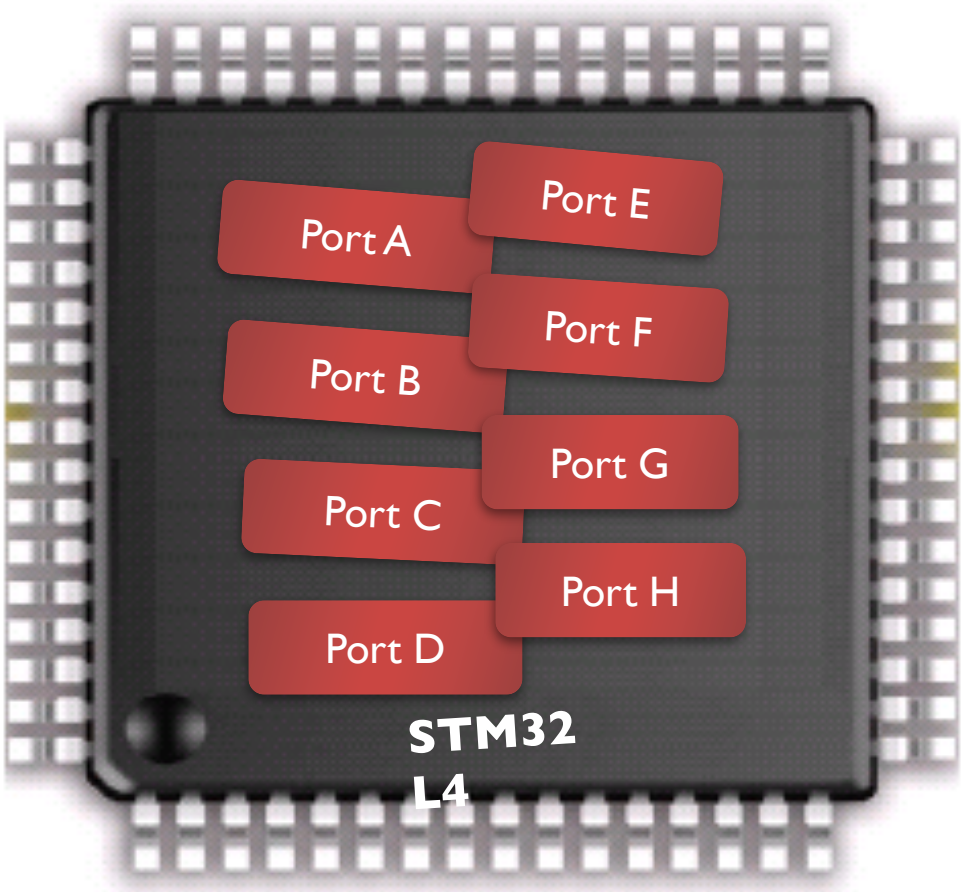
MTB: Micro Trace Buffer



Memory Map of STM32L4



General Purpose Input/Output (GPIO)

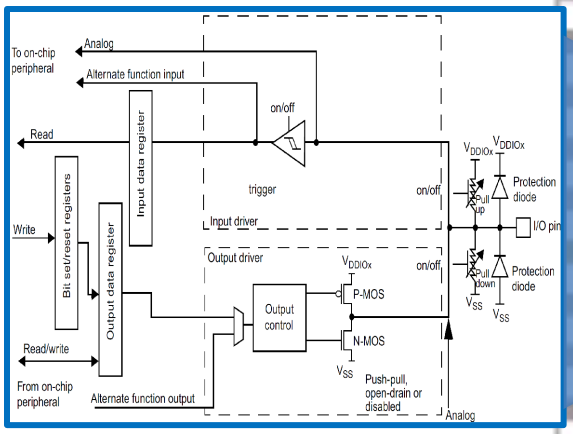


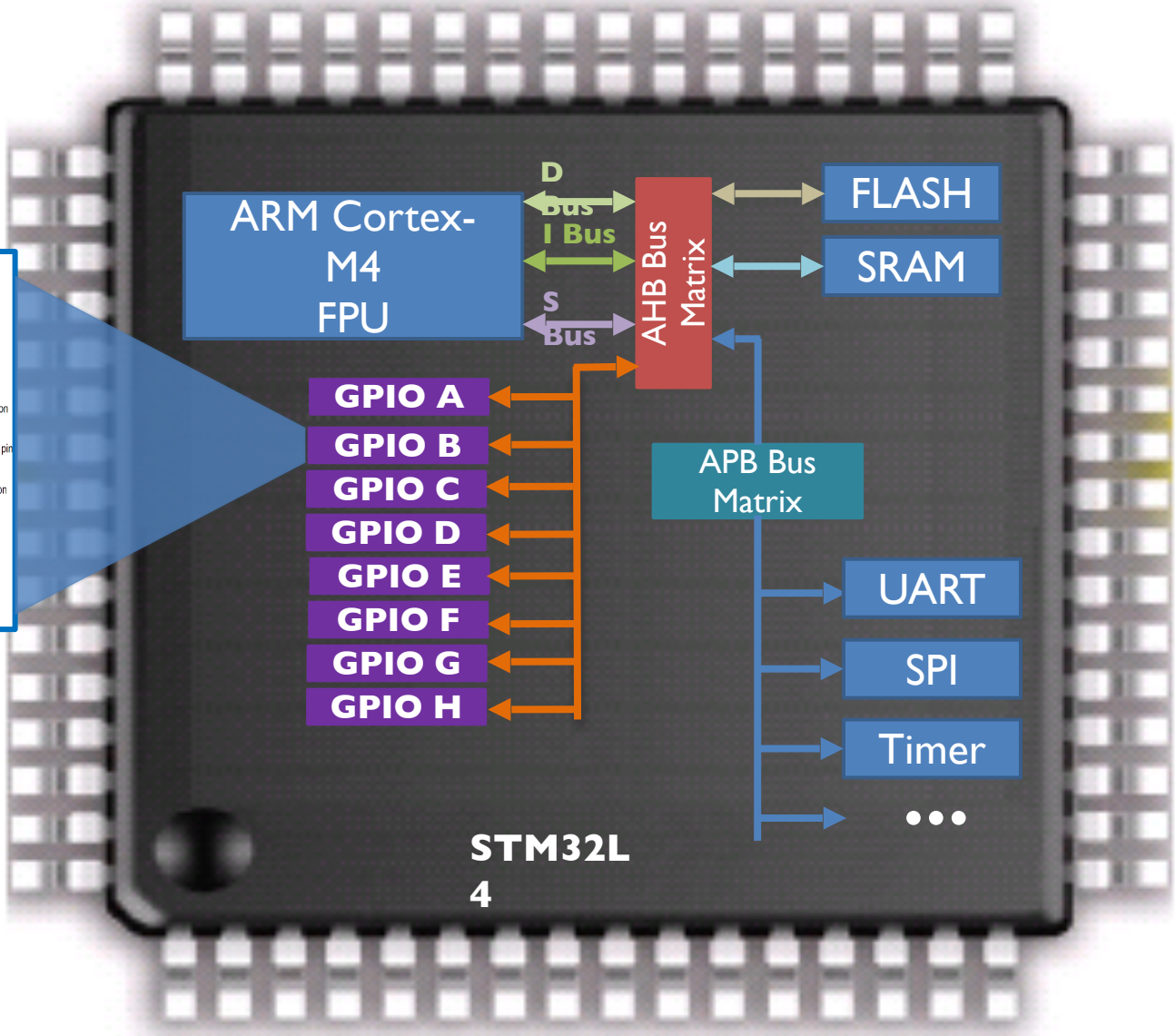
8 GPIO Ports:

A, B, C, D, E, F, G, H

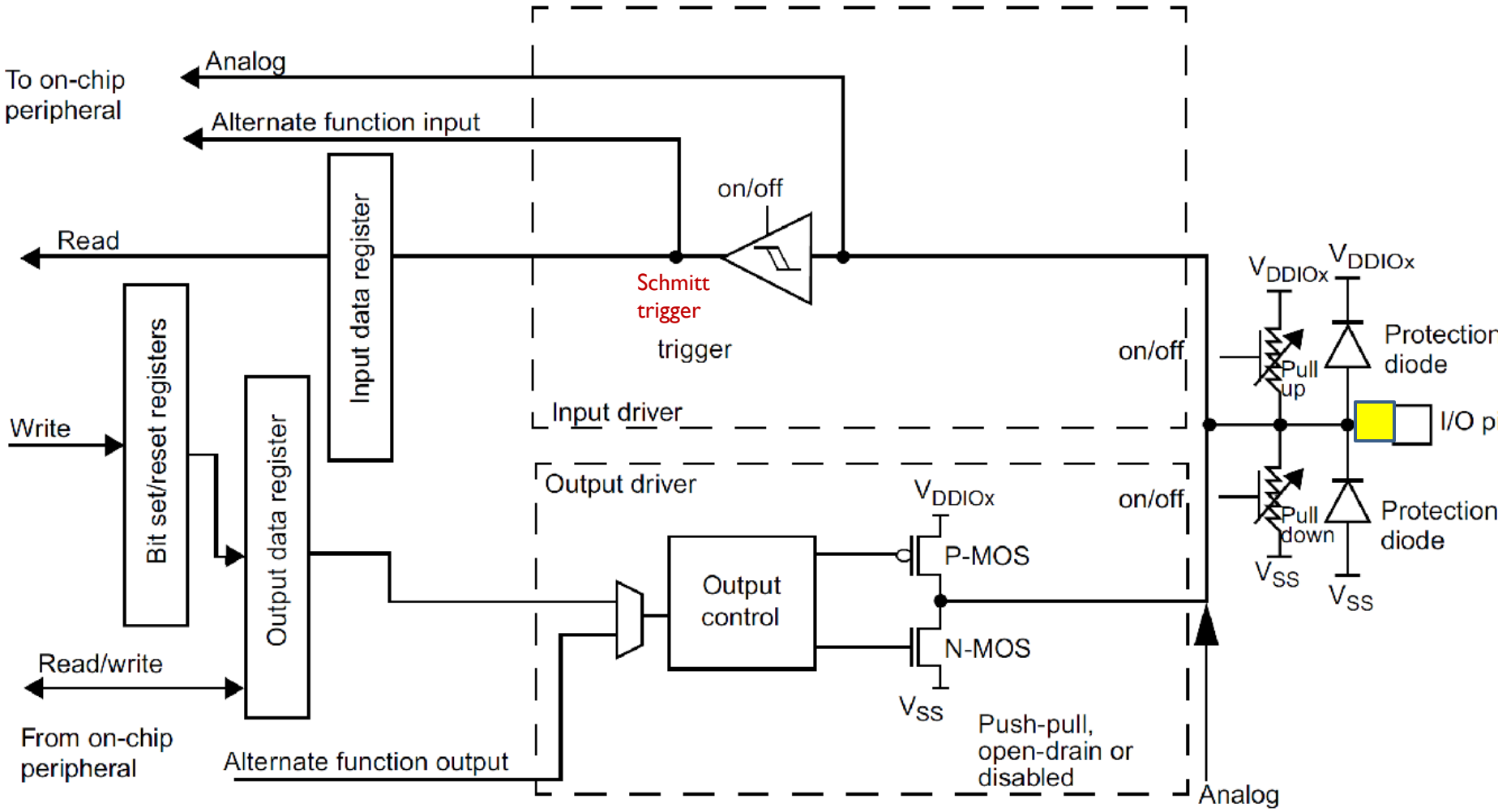
Up to 16 pins in each port

General Purpose Input/Output (GPIO)

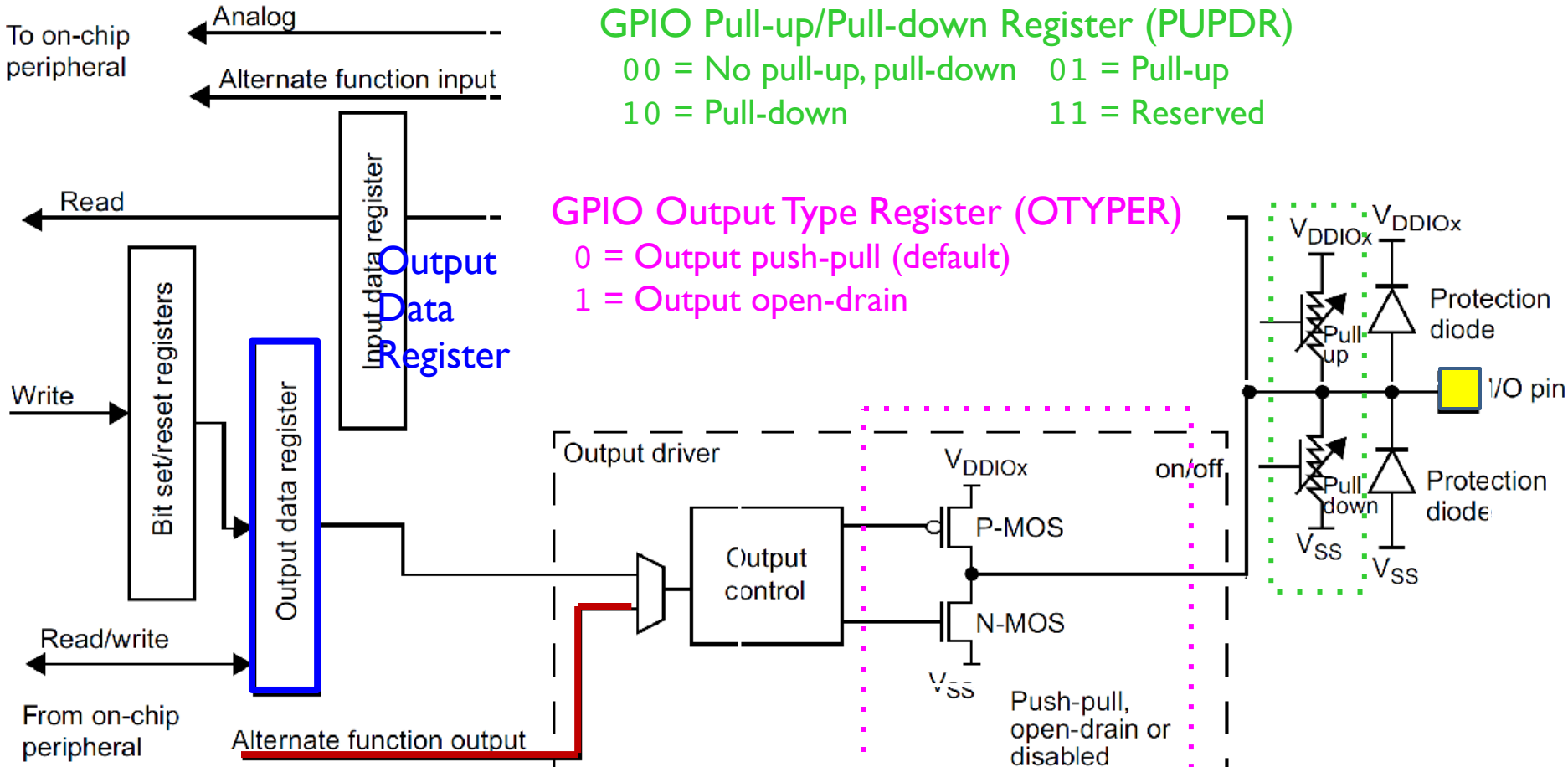




Basic Structure of an I/O Port Bit
Input and Output



Basic Structure of an I/O Port Bit: Output



GPIO Pull-up/Pull-down Register (PUPDR)

00 = No pull-up, pull-down 01 = Pull-up

10 = Pull-down 11 = Reserved

GPIO Output Type Register (OTYPER)

0 = Output push-pull (default)

1 = Output open-drain

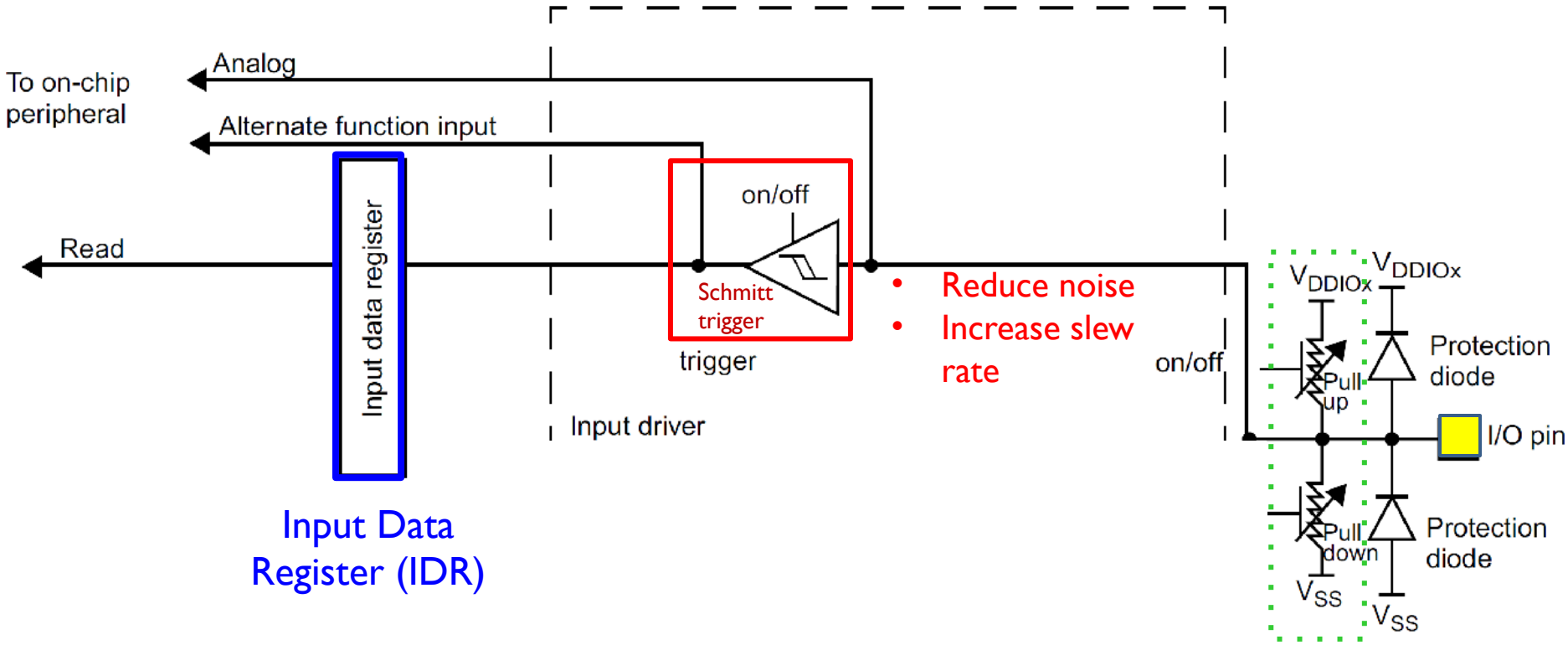
GPIO MODE Register

00 = Input, 01 = Output,

10 = AF, 11 = Analog

(default)

Basic Structure of an I/O Port Bit: Input



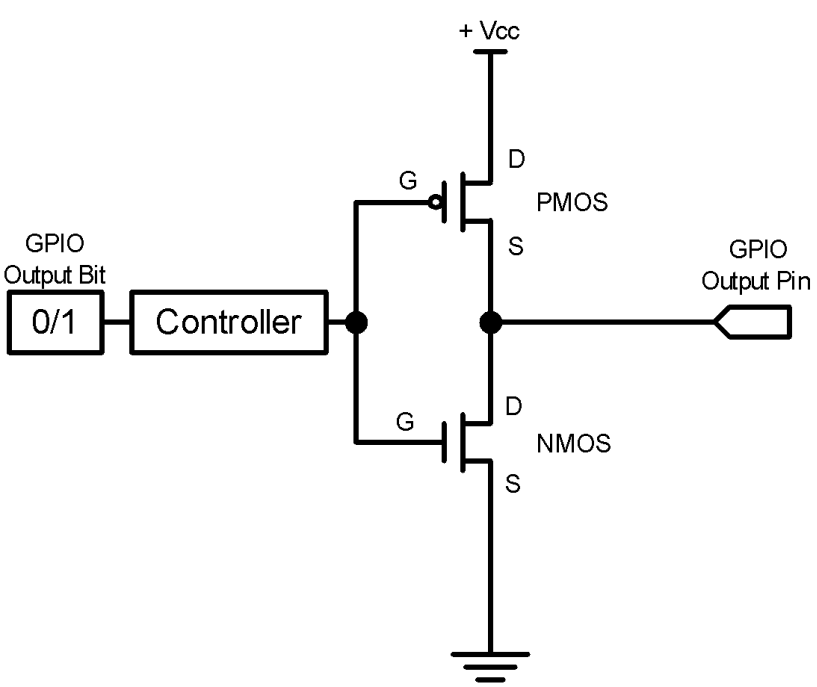
Input Data Register (IDR)

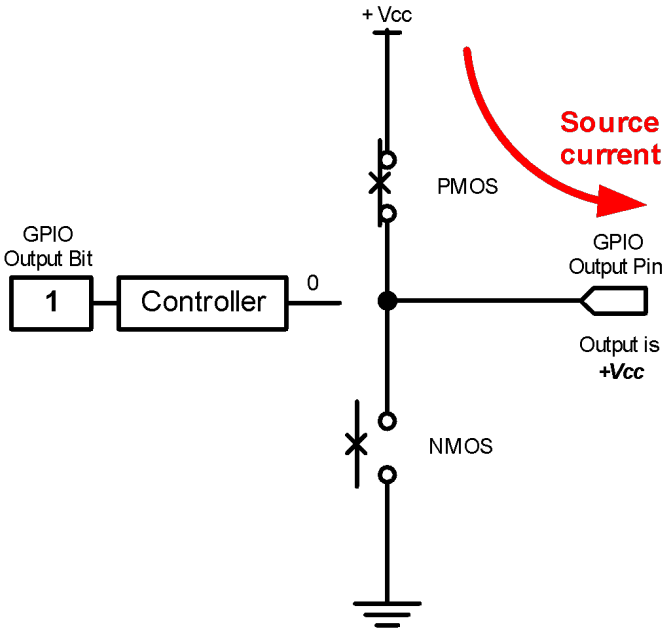
GPIO Pull-up/Pull-down Register (PUPDR)

00 = No pull-up, pull-down 01 = Pull-up

10 = Pull-down 11 = Reserved

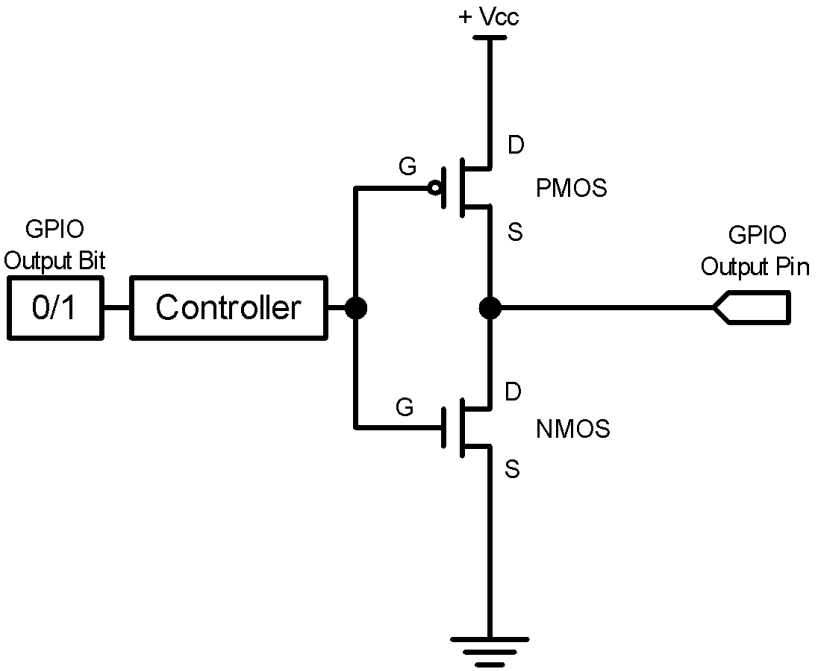
GPIO Output: Push-Pull

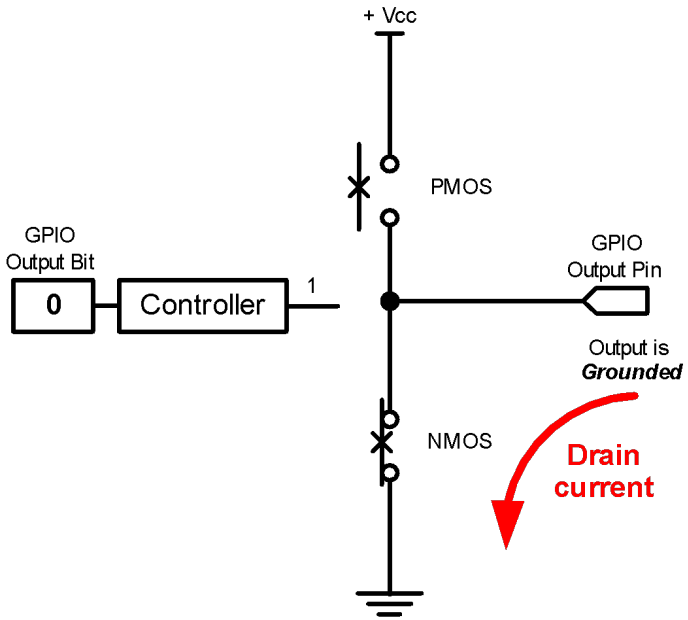




GPIO Output = 1
Source current to external circuit

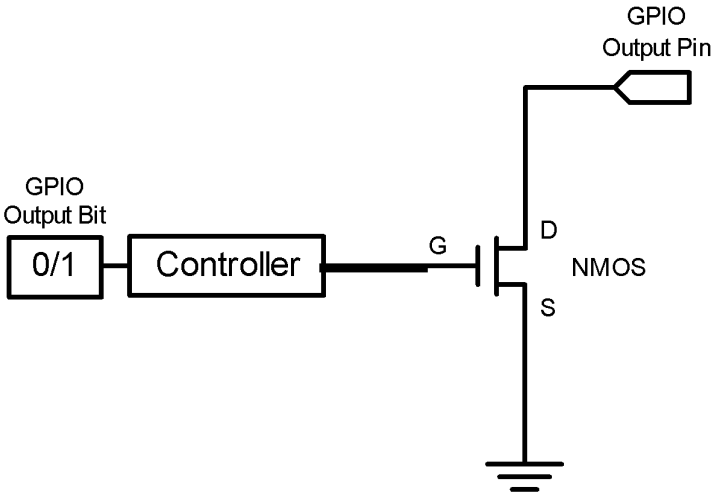
GPIO Output: Push-Pull

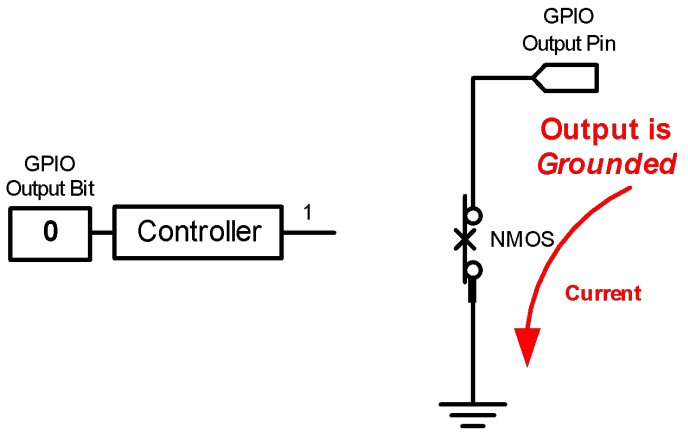




GPIO Output = 0
Drain current from external circuit

GPIO Output: Open-Drain

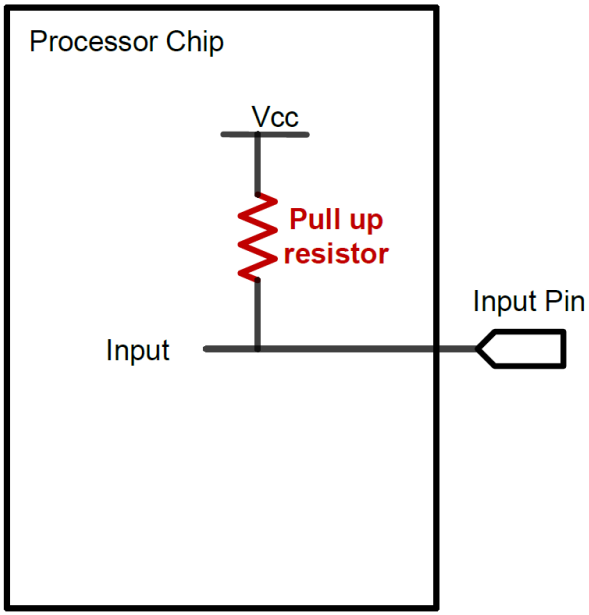




GPIO Output = 0
Drain current from external circuit

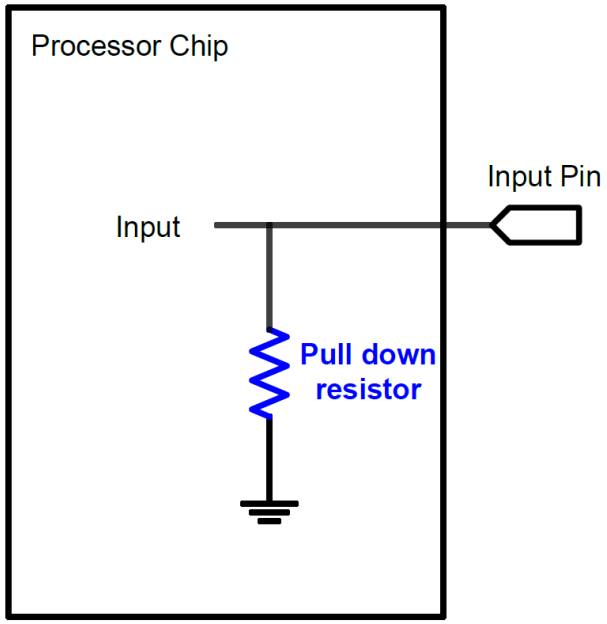
GPIO Input: Pull Up and Pull Down

- A digital input can have three states: High, Low, and High-Impedance (also called floating, tri-stated, HiZ)



Pull-Up

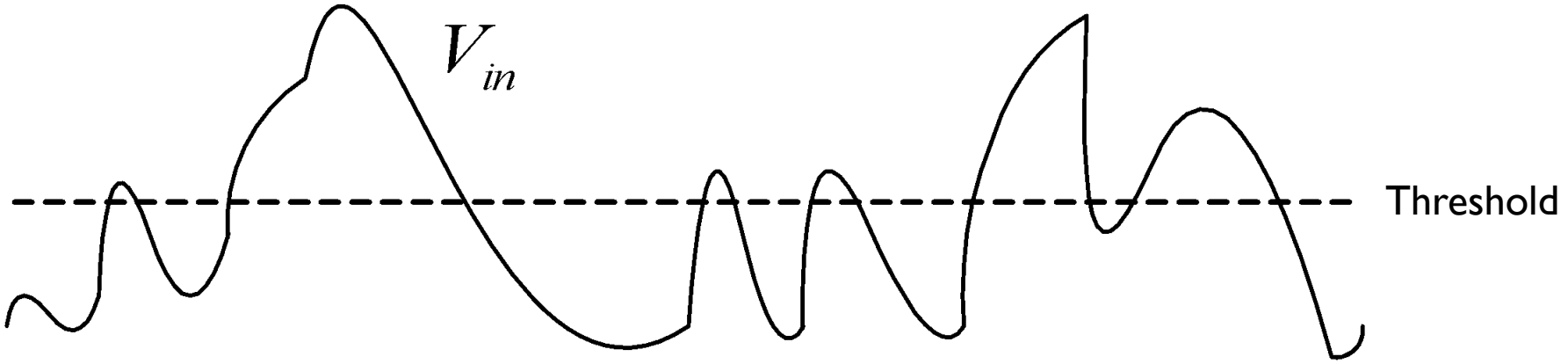
If external input is HiZ, the input is read as a valid HIGH.



Pull-Down

If external input is HiZ, the input is read as a valid LOW.

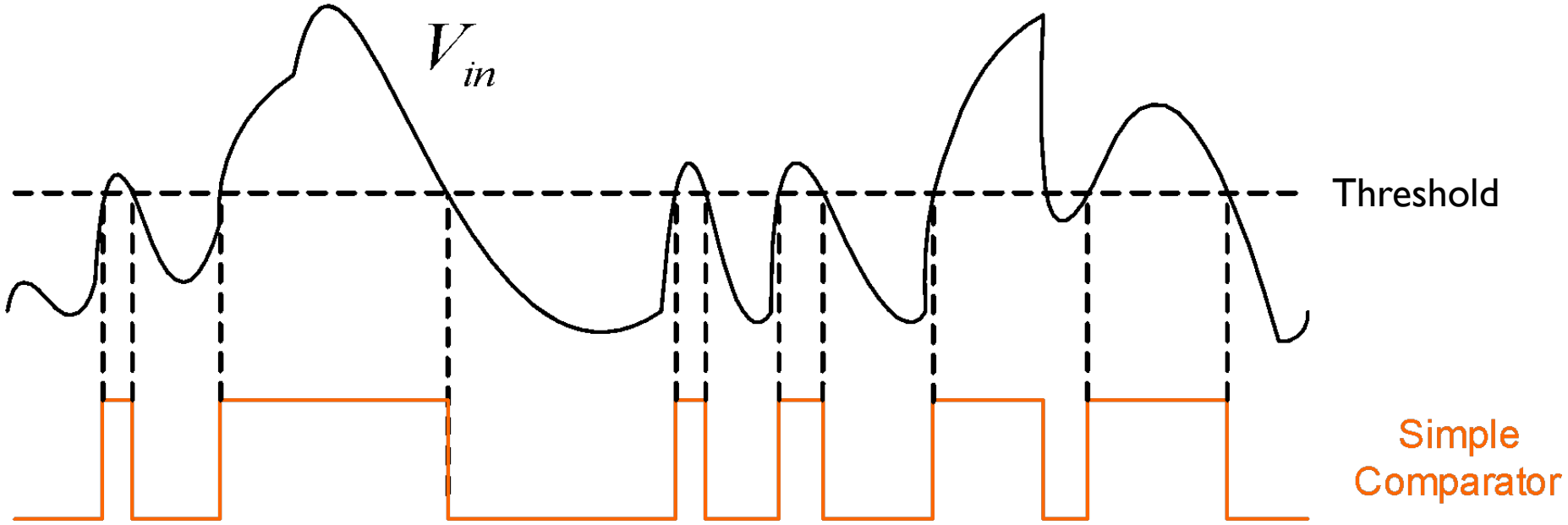
Schmitt Trigger



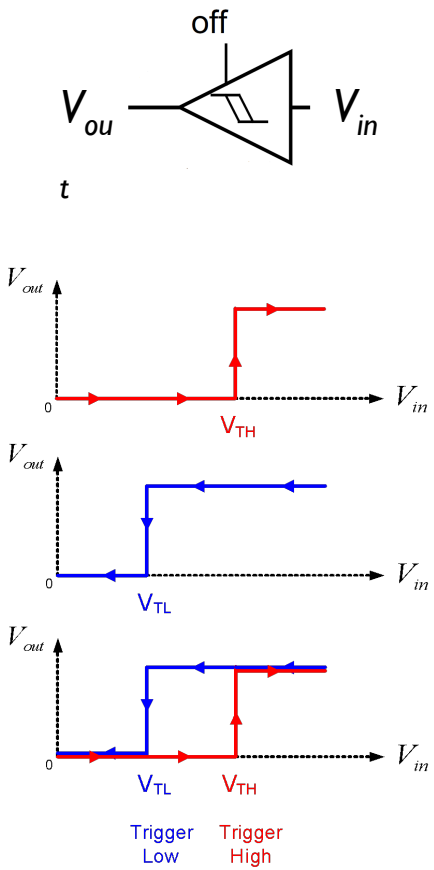
Analog signals

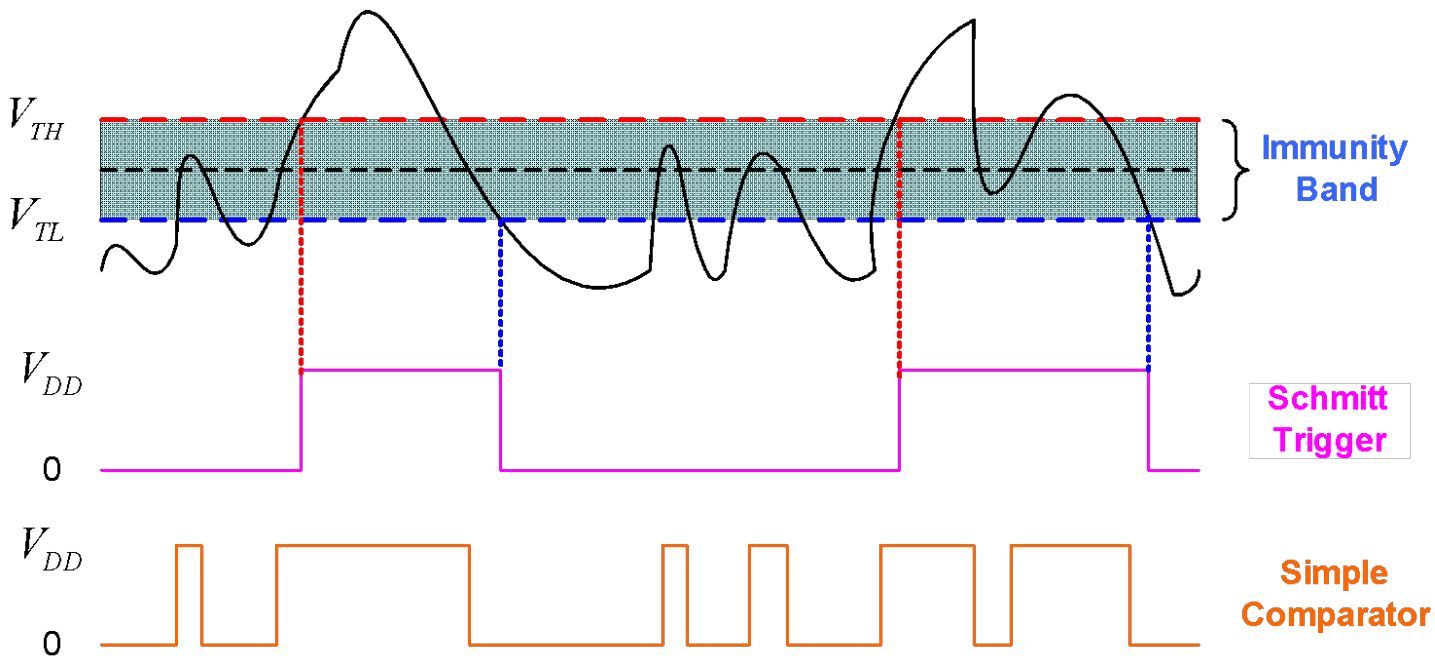
- Noisy
- Rise and fall slowly (small slew rate)

Schmitt Trigger

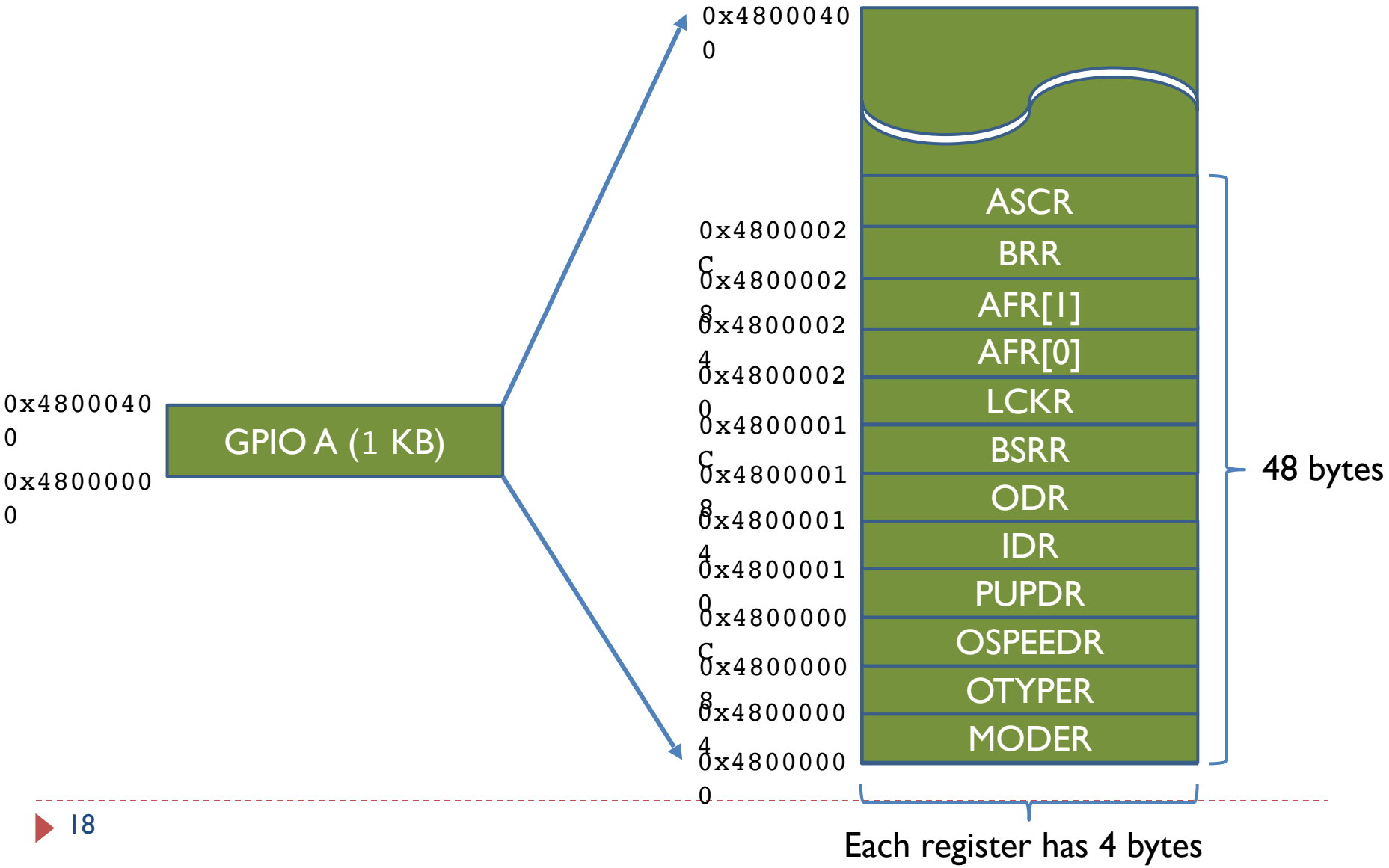


Schmitt Trigger

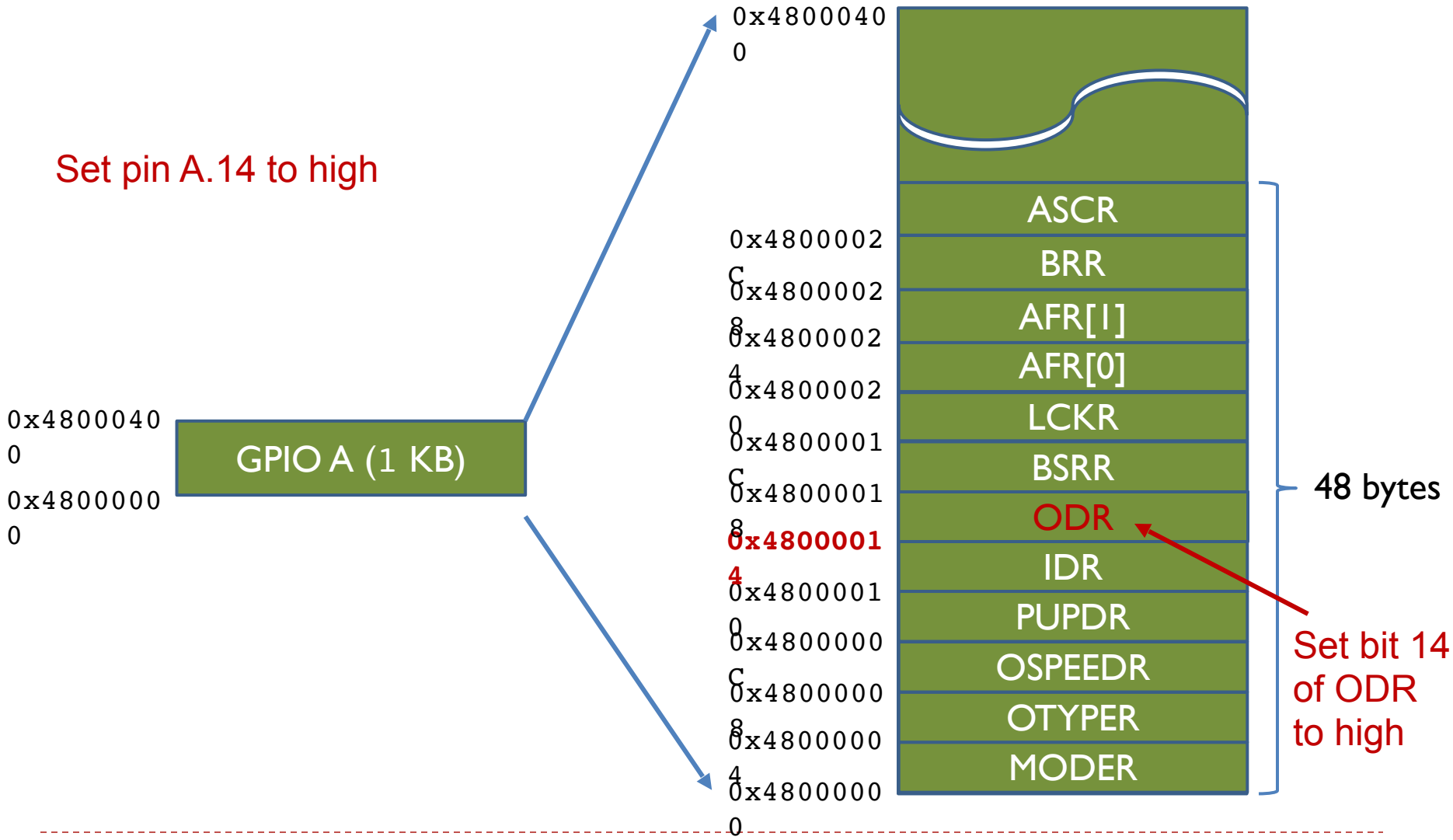




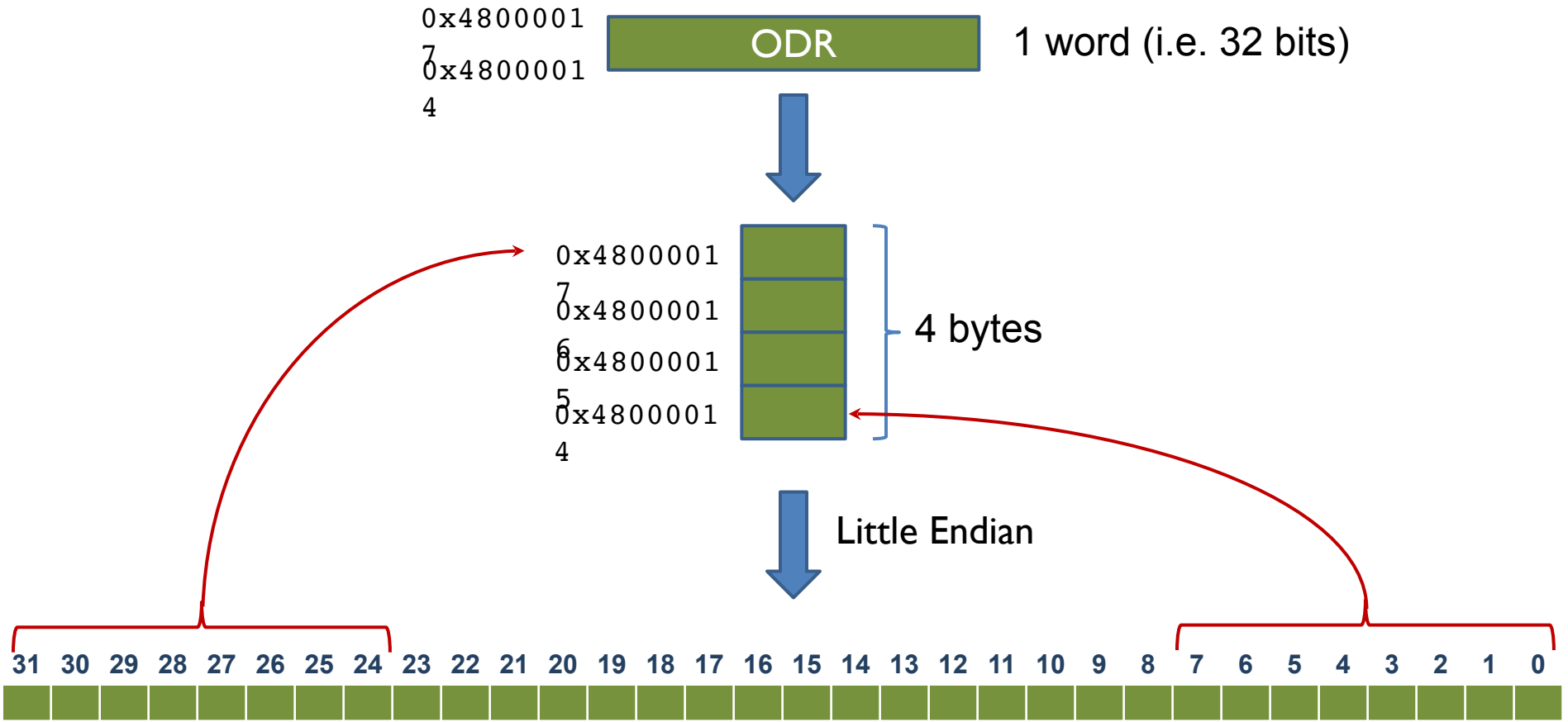
GPIO Memory Map



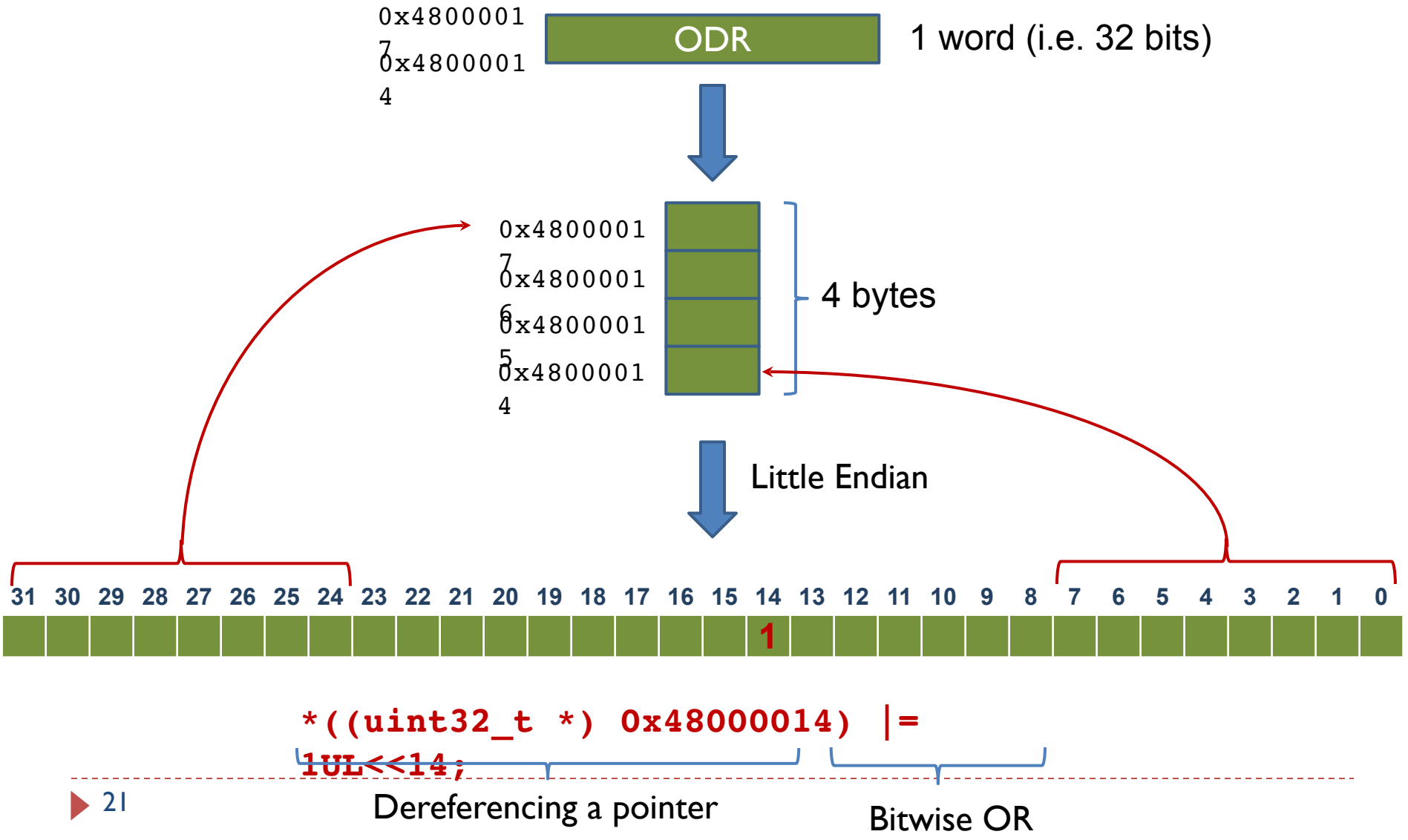
GPIO Memory Map



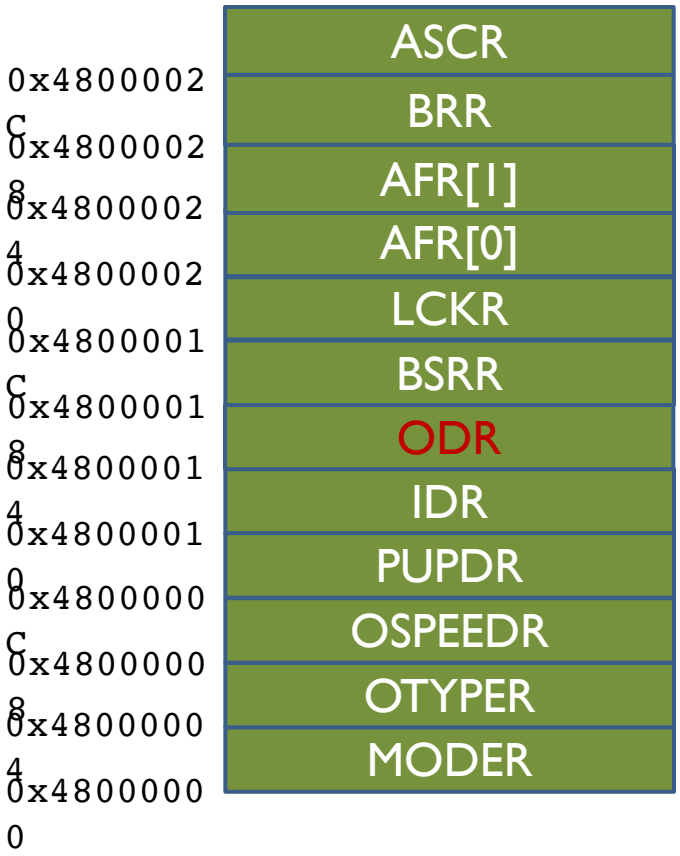
Output Data Register (ODR)



Output Data Register (ODR)



Dereferencing a Memory Address



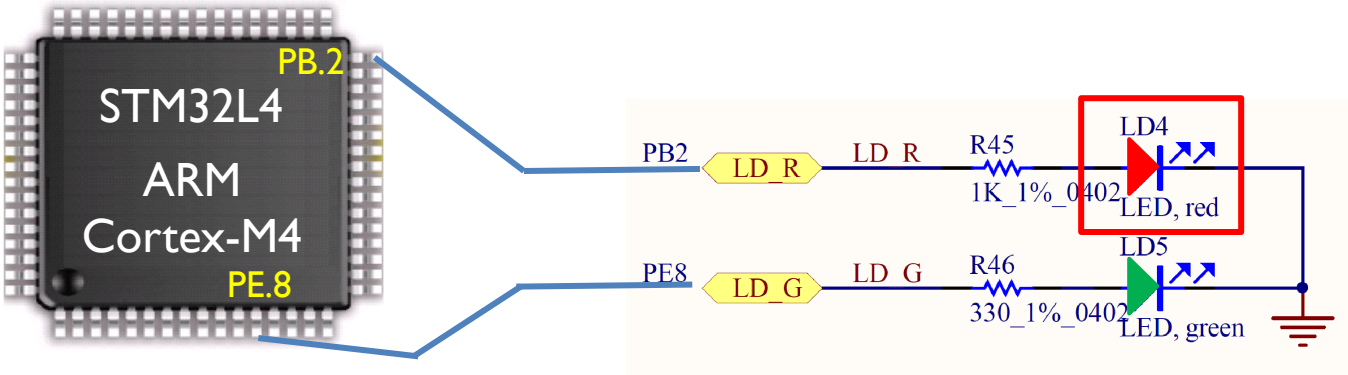

```
typedef struct {
  volatile uint32_t MODER;    // Mode register
  volatile uint32_t OTYPER;   // Output type register
  volatile uint32_t OSPEEDR;  // Output speed register
  volatile uint32_t PUPDR;    // Pull-up/pull-down
register
  volatile uint32_t IDR;      // Input data register
  volatile uint32_t ODR;      // Output data register
  volatile uint32_t BSRR;     // Bit set/reset register
  volatile uint32_t LCKR;     // Configuration lock
register
  volatile uint32_t AFR[2];   // Alternate function
registers
  volatile uint32_t BRR;      // Bit Reset register
  volatile uint32_t ASCR;     // Analog switch control
register
} GPIO_TypeDef;

// Casting memory address to a pointer
#define GPIOA ((GPIO_TypeDef *) 0x48000000)
```

```
GPIOA->ODR |= 1UL<<14;
```

or

```
(*GPIOA).ODR |= 1UL<<14;
```

Red LED (PB.2)



PB.2	Red LED
High	On
Low	Off

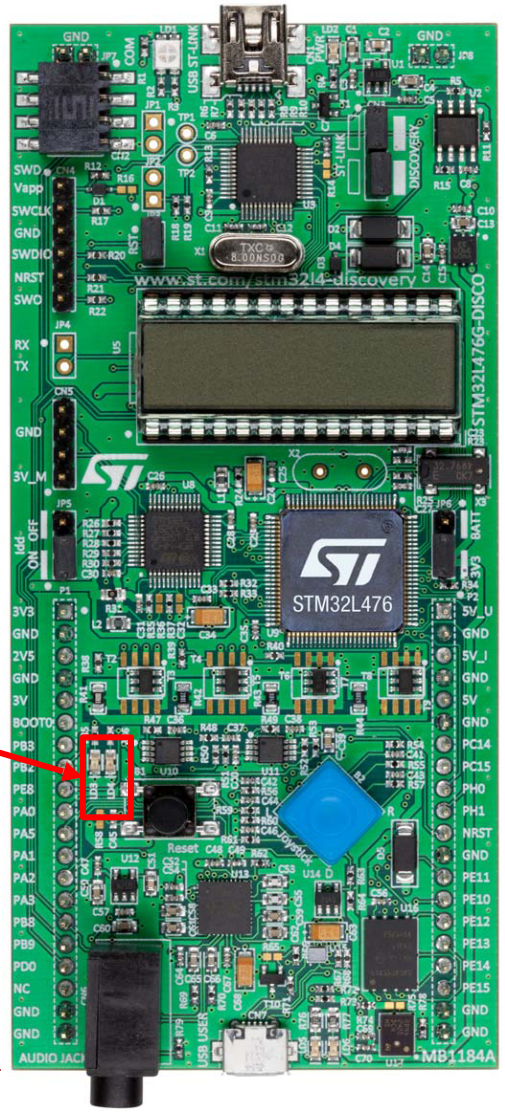


Figure 14-18 shows the flowchart of initializing a GPIO pin as digital output with push-pull.

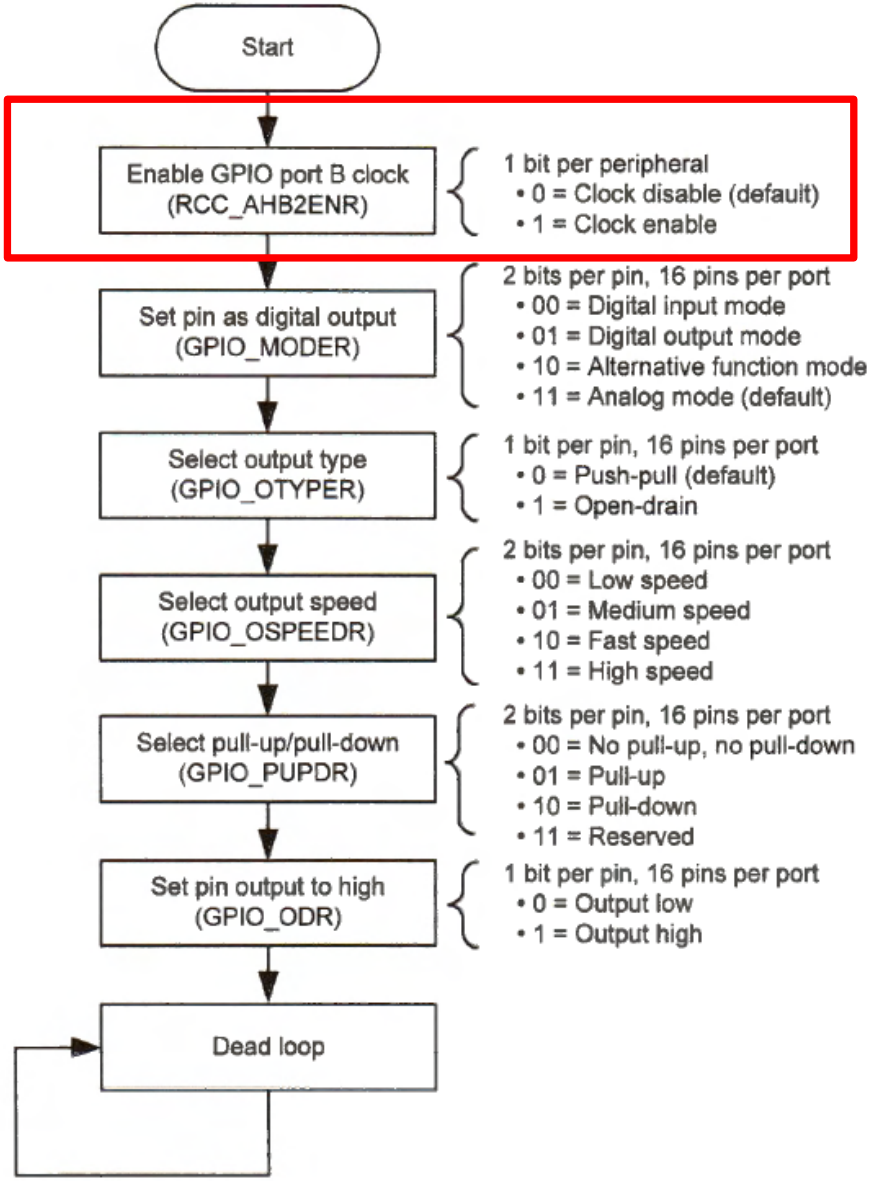


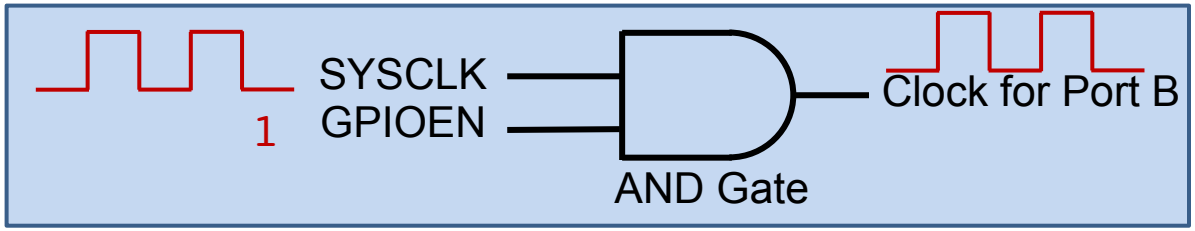
Figure 14-18. Flowchart of GPIO initialization

Enable Clock

- AHB2 peripheral clock enable register (RCC_AHB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RNG EN	Res.	AESEN
													rw		rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	ADCEN	OTGFS EN	Res.	Res.	Res.	Res.	GPIOH EN	GPIOG EN	GPIOF EN	GPIOE EN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN
		rw	rw					rw	rw	rw	rw	rw	rw	rw	rw

Bit 1 **GPIOBEN:** IO port B clock enable
Set and cleared by software.
0: IO port B clock disabled
1: IO port B clock enabled




```
#define  RCC_AHB2ENR_GPIOBEN  ((uint32_t)0x00000002U)

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

Figure 14-18 shows the flowchart of initializing a GPIO pin as digital output with push-pull.

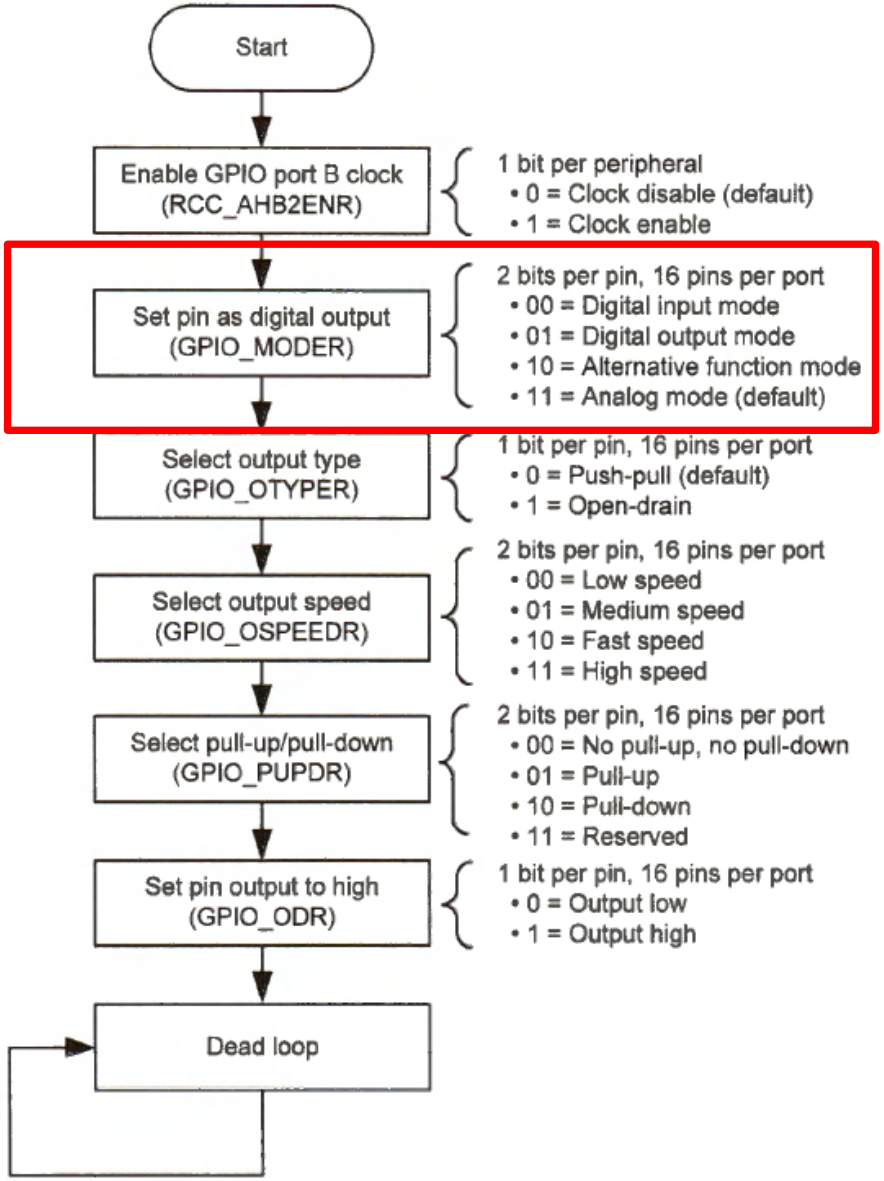


Figure 14-18. Flowchart of GPIO initialization

GPIO Mode Register (MODER)

- 32 bits (16 pins, 2 bits per pin)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODE15[1:0]		MODE14[1:0]		MODE13[1:0]		MODE12[1:0]		MODE11[1:0]		MODE10[1:0]		MODE9[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODE7[1:0]		MODE6[1:0]		MODE5[1:0]		MODE4[1:0]		MODE3[1:0]		MODE2[1:0]		MODE1[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Pin 2 Pin 1 Pin 0

Bits 2y+1:2y **MODEy[1:0]:** Port x configuration bits (y = 0..15)

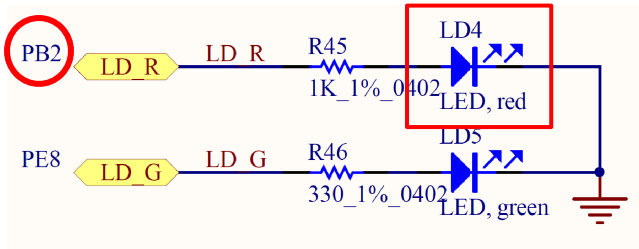
These bits are written by software to configure the I/O mode.

00: Input mode

01: General purpose output mode

10: Alternate function mode

11: Analog mode (reset state)




```
GPIOB->MODER &= ~(3UL<<4);   // Clear bits 4 and 5 for Pin 2
GPIOB->MODER |=   1UL<<4;    // Set bit 4, set Pin 2 as output
```

Figure 14-18 shows the flowchart of initializing a GPIO pin as digital output with push-pull.

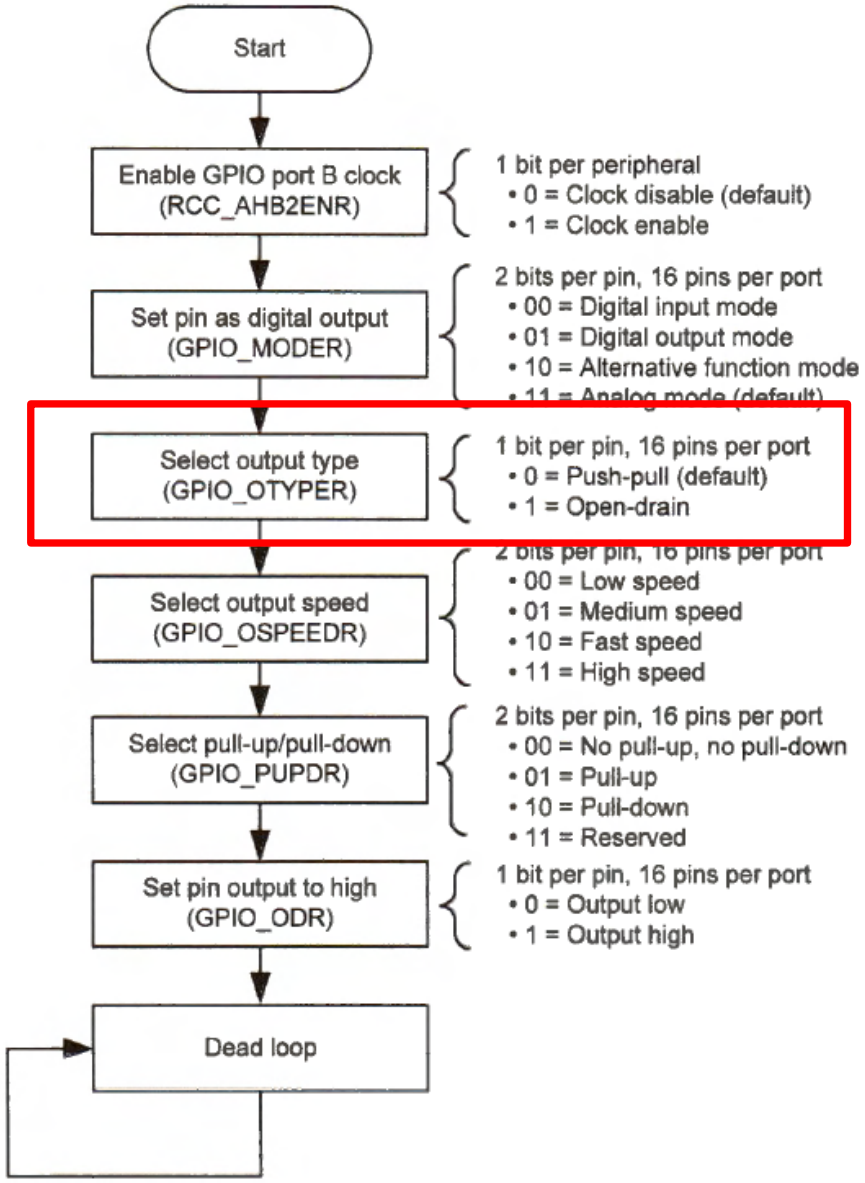


Figure 14-18. Flowchart of GPIO initialization

GPIO Output Type Register (OTYPE)

- 16 bits reserved, 16 data bits, 1 bit for each pin

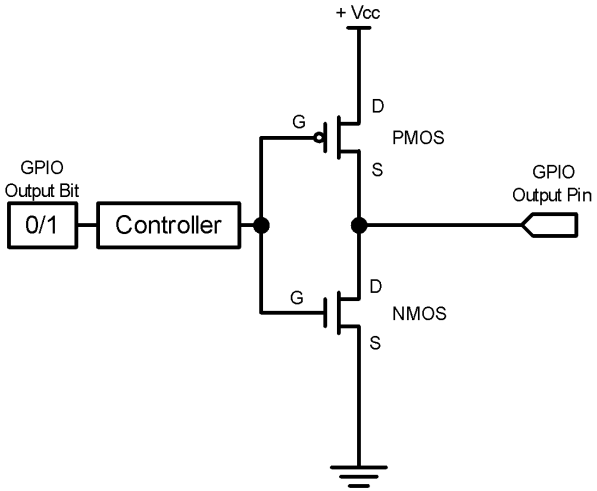
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **OTy:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output type.

0: Output push-pull (reset state)

1: Output open-drain




```
GPIOB->OTYPE &= ~(1UL<<2);  // Clear
bit 2
```

Figure 14-18 shows the flowchart of initializing a GPIO pin as digital output with push-pull.

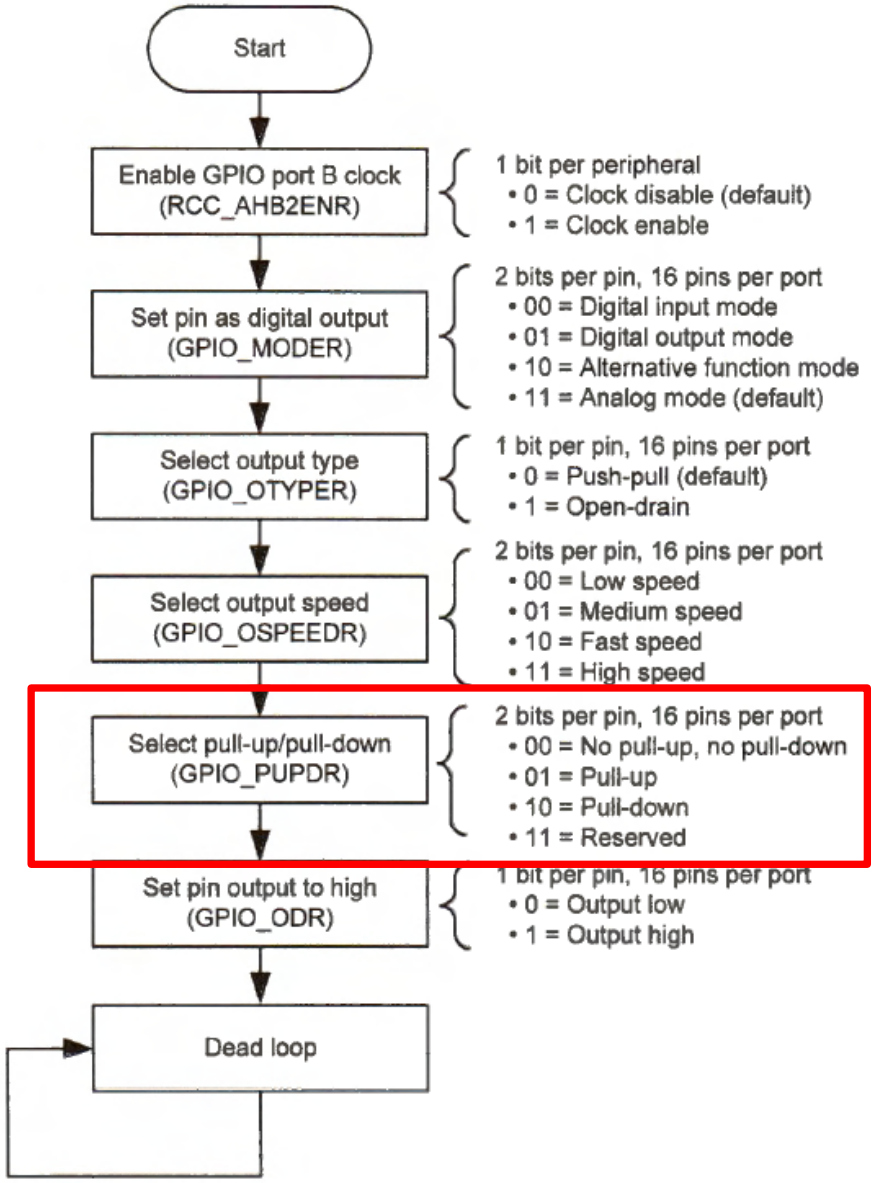


Figure 14-18. Flowchart of GPIO initialization

GPIO Pull-up/Pull-down Register (PUPDR)

- 16 pins per port, 2 bits per pin

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPD15[1:0]		PUPD14[1:0]		PUPD13[1:0]		PUPD12[1:0]		PUPD11[1:0]		PUPD10[1:0]		PUPD9[1:0]		PUPD8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPD7[1:0]		PUPD6[1:0]		PUPD5[1:0]		PUPD4[1:0]		PUPD3[1:0]		PUPD2[1:0]		PUPD1[1:0]		PUPD0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

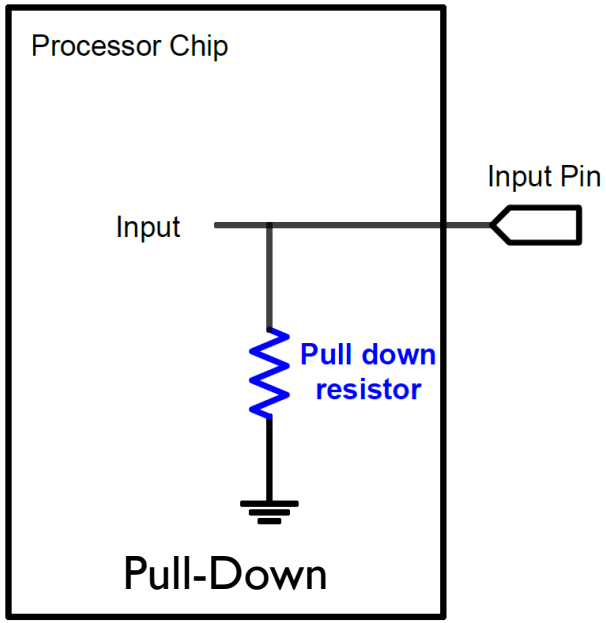
Pin 2 Pin 1 Pin 0

Bits 2y+1:2y **PUPDy[1:0]:** Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

00: No pull-up, pull-down
01: Pull-up
10: Pull-down
11: Reserved

```
// No pull-up, pull-down
GPIOA->PUPDR &= ~3UL;
```



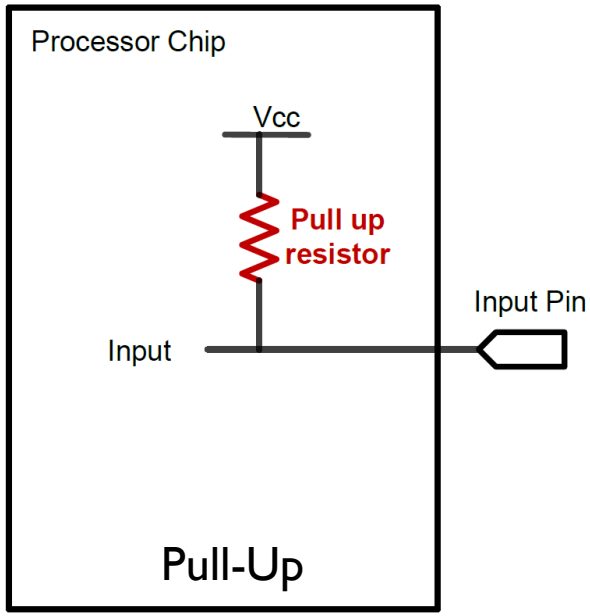


Figure 14-18 shows the flowchart of initializing a GPIO pin as digital output with push-pull.

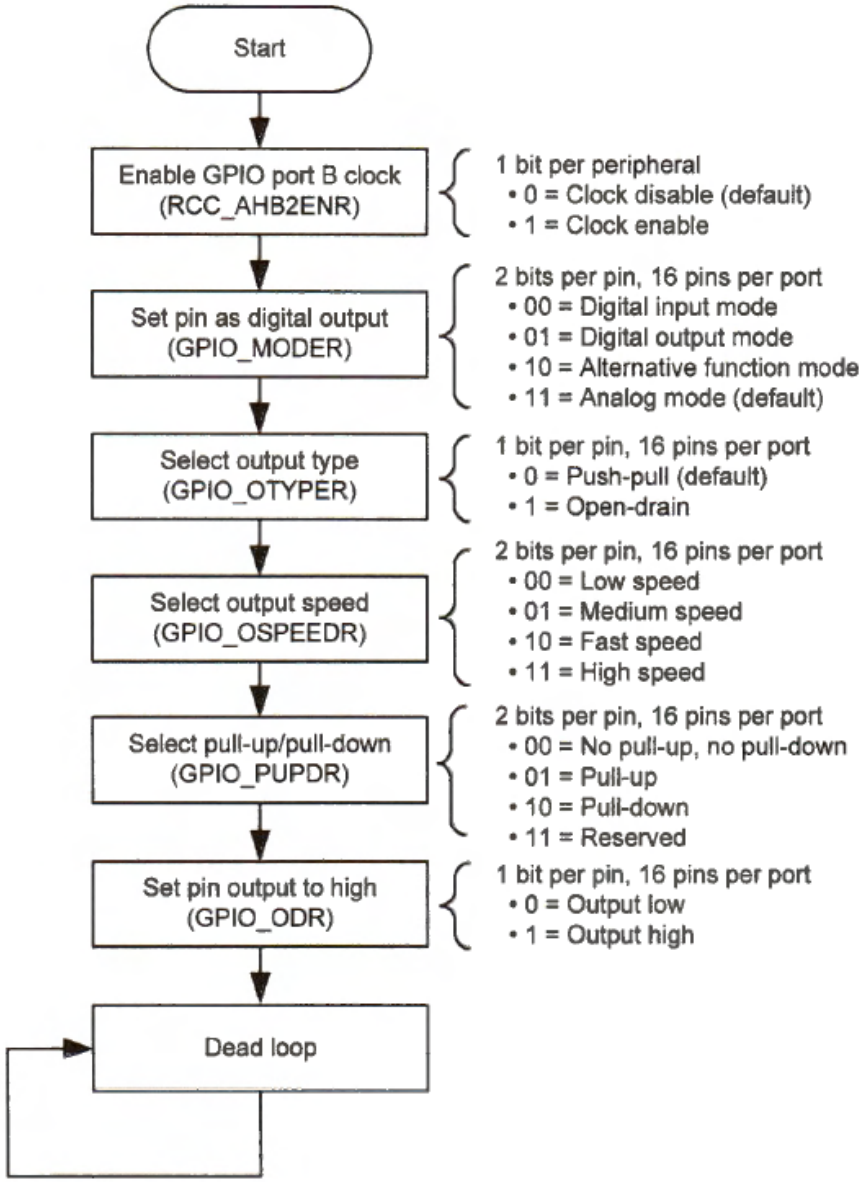


Figure 14-18. Flowchart of GPIO initialization

```
// Reset and clock control
typedef struct {
  __IO uint32_t CR;         // Clock control register
  __IO uint32_t ICSCR;      // Internal clock sources calibration register
  __IO uint32_t CFGR;       // Clock configuration register
  ...
  __IO uint32_t AHB1ENR;    // AHB1 peripheral clocks enable register
  __IO uint32_t AHB2ENR;    // AHB2 peripheral clocks enable register
  __IO uint32_t AHB3ENR;    // AHB3 peripheral clocks enable register
  ...
} RCC_TypeDef;

#define RCC ((RCC_TypeDef *) 0x40021000))
```

```
#define RCC_AHB2ENR_GPIOBEN  (0x00000002)

RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
```

```c
// Red LED is connected PB 2 (GPIO port B pin 2)
void GPIO_Clock_Enable(){
    // Enable the clock to GPIO port B
    RCC->AHB2ENR |= RCC_AHB2ENR_GPIOBEN;
}

void GPIO_Pin_Init(){

    // Set mode of pin 2 as digital output
    // 00 = digital input,          01 = digital output
    // 10 = alternate function,     11 = analog (default)
    GPIOB->MODER &= ~(3UL<<4);    // Clear mode bits
    GPIOB->MODER |= 1UL<<4;       // mode = 01, digital output

    // Set output type of pin 2 as push-pull
    // 0 = push-pull (default)
    // 1 = open-drain
    GPIOB->OTYPER &= ~(1<<2);

    // Set output speed of pin 2 as low
    // 00 = Low speed,     01 = Medium speed
    // 10 = Fast speed,    11 = High speed
    GPIOB->OSPEEDR &= ~(3UL<<4);      // Clear speed bits

    // Set pin 2 as no pull-up, no pull-down
    // 00 = no pull-up, no pull-down  01 = pull-up
    // 10 = pull-down,                11 = reserved
    GPIOB->PUPDR &= ~(3UL<<4);         // no pull-up, no pull-down
}
```

```
int main(void){
    GPIO_Clock_Enable();
    GPIO_Pin_Init();
    GPIOB->ODR |= 1UL<<6;   // Set bit 6 of output data register (ODR)
    while(1);               // Dead loop & program hangs here
}
```

User

It should be 1UL<< 2; to set the bit 2

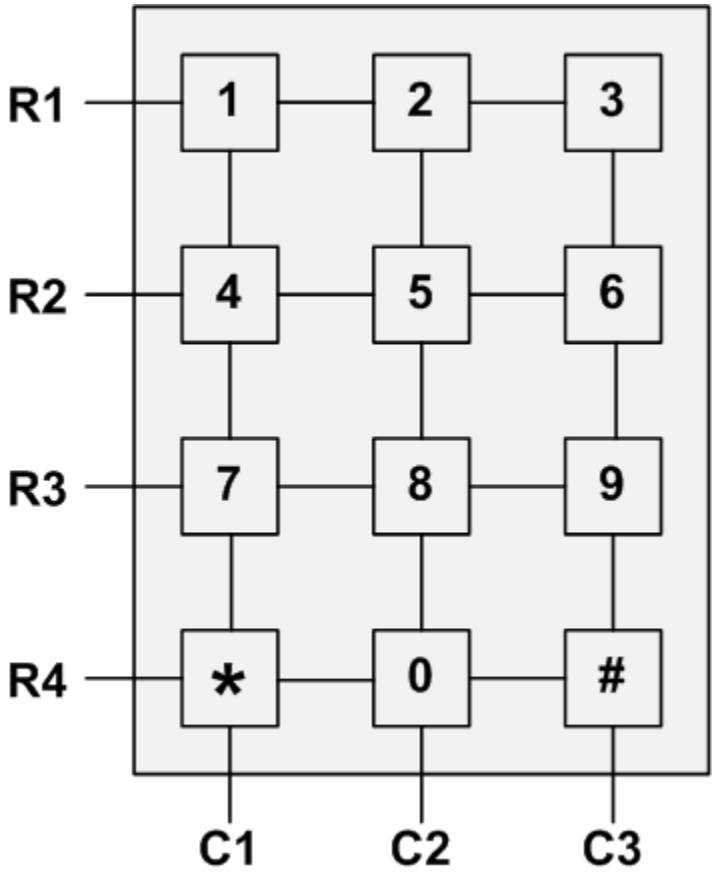
e 14-1. Lighting up an LED in C

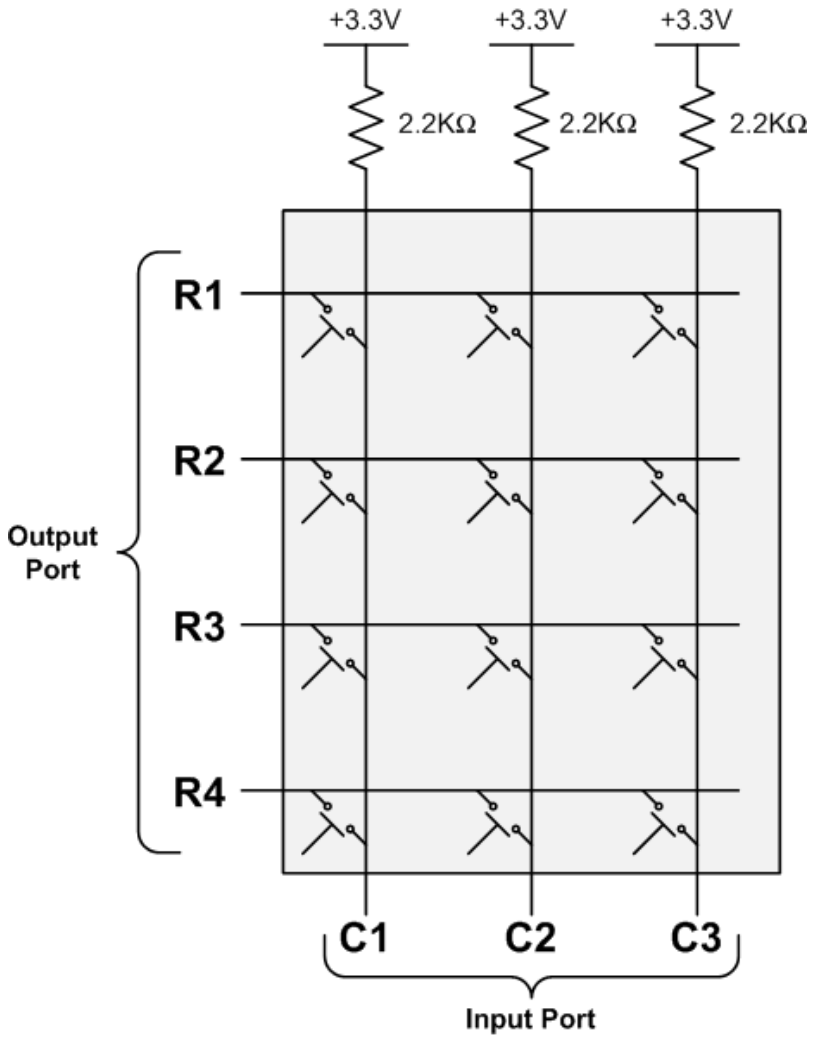
```
; Constants defined in file stm32l476xx_constants.s
;
; Memory addresses of GPIO port B and RCC (reset and clock control) data
; structure. These addresses are predefined by the chip manufacturer.
GPIOB_BASE          EQU   0x48000400
RCC_BASE            EQU   0x40021000

; Byte offset of each variable in the GPIO_TypeDef structure
GPIO_MODER          EQU   0x00
GPIO_OTYPER         EQU   0x04
GPIO_RESERVED0      EQU   0x06
GPIO_OSPEEDR        EQU   0x08
GPIO_PUPDR          EQU   0x0C
GPIO_IDR            EQU   0x10
GPIO_RESERVED1      EQU   0x12
GPIO_ODR            EQU   0x14
GPIO_RESERVED2      EQU   0x16
GPIO_BSRRL          EQU   0x18
GPIO_BSRRH          EQU   0x1A
GPIO_LCKR           EQU   0x1C
GPIO_AFR0           EQU   0x20  ;  AFR[0]
GPIO_AFR1           EQU   0x24  ;  AFR[1]
GPIO_AFRL           EQU   0x20
GPIO_AFRH           EQU   0x24

; Byte offset of variable AHB2ENR in the RCC_TypeDef structure
RCC_AHB2ENR         EQU   0x4C
```

```
  INCLUDE stm32l476xx_constants.s

  AREA    main, CODE, READONLY
  EXPORT      __main                  ; make __main visible to linker
  ENTRY

__main PROC
  ; Enable the clock to GPIO port B
  ; Load address of reset and clock control (RCC)
  LDR r2, =RCC_BASE                   ; Pseduo instruction
  LDR r1, [r2, #RCC_AHB2ENR]          ; r1 = RCC->AHB2ENR
  ORR r1, r1, #2                      ; Set bit 2 of AHB2ENR
  STR r1, [r2, #RCC_AHB2ENR]          ; GPIO port B clock enable

  ; Load GPIO port B base address
  LDR r3, =GPIOB_BASE                 ; Pseudo instruction

  ; Set pin 2 I/O mode as general-purpose output
  LDR r1, [r3, #GPIO_MODER]           ; Read the mode register
  BIC r1, r1, #(3 << 4)               ; Direction mask pin 6, clear bits 5 and 4
  ORR r1, r1, #(1 << 4)               ; Set mode as digital output (mode = 01)
  STR r1, [r3, #GPIO_MODER]           ; Save to the mode register
```

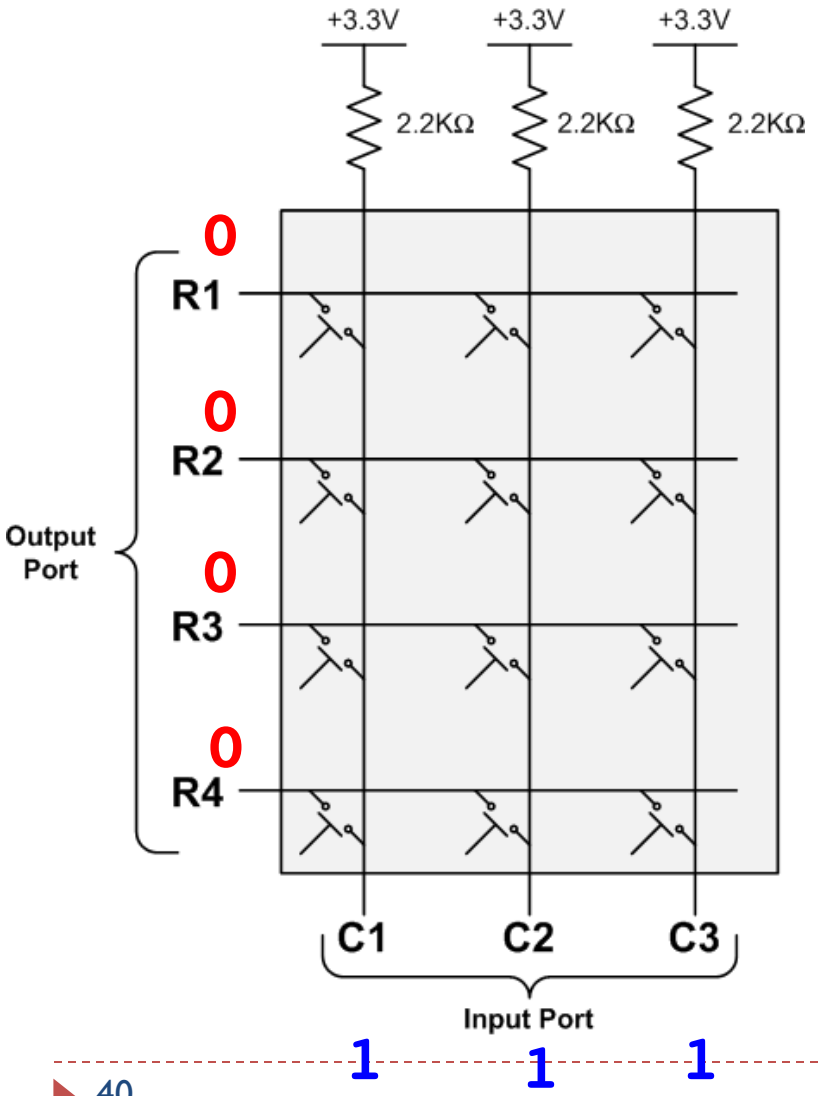
```
  ; Set pin 2 the push-pull mode for the output type
  LDR r1, [r3, #GPIO_OTYPER]    ; Read the output type register
  BIC r1, r1, #(1<<2)           ; Push-pull(0), open-drain (1)
  STR r1, [r3, #GPIO_OTYPER]    ; Save to the output type register

  ; Set I/O output speed value as low
  LDR r1, [r3, #GPIO_OSPEEDR]   ; Read the output speed register
  BIC r1, r1, #(3<<4)           ; Low(00), Medium(01), Fast(01), High(11)
  STR r1, [r3, #GPIO_OSPEEDR]   ; Save to the output speed register

  ; Set I/O as no pull-up, no pull-down
  LDR r1, [r3, #GPIO_PUPDR]     ; r1 = GPIOB->PUPDR
  BIC r1, r1, #(3<<4)           ; No PUPD(00), PU(01), PD(10), Reserved(11)
  STR r1, [r3, #GPIO_PUPDR]     ; Save pull-up and pull-down setting

  ; Light up LED
  LDR r1, [r3, #GPIO_ODR]       ; Read the output data register
  ORR r1, r1, #(1<<2)           ; Set bit 2
  STR r1, [r3, #GPIO_ODR]       ; Save to the output data register

stop
  B  stop  ; dead loop & program hangs here
  ENDP
  END
```

Keypad Scan


R1
R2
R3
R4
1
2
3
4
5
6
7
8
9
*
0
#
C1
C2
C3


+3.3V
+3.3V
+3.3V
2.2KΩ
2.2KΩ
2.2KΩ
R1
R2
R3
R4
Output Port
C1
C2
C3
Input Port

Keypad Scan



Step 1: Set Output

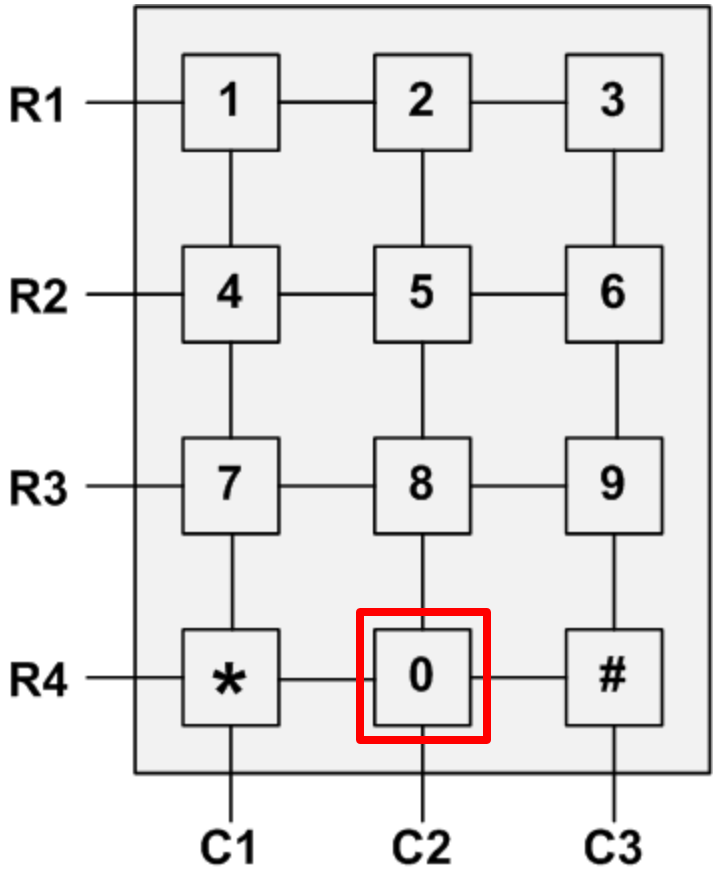
```
R1,R2,R3,R4 = 0000
```

Step 2: Read Input

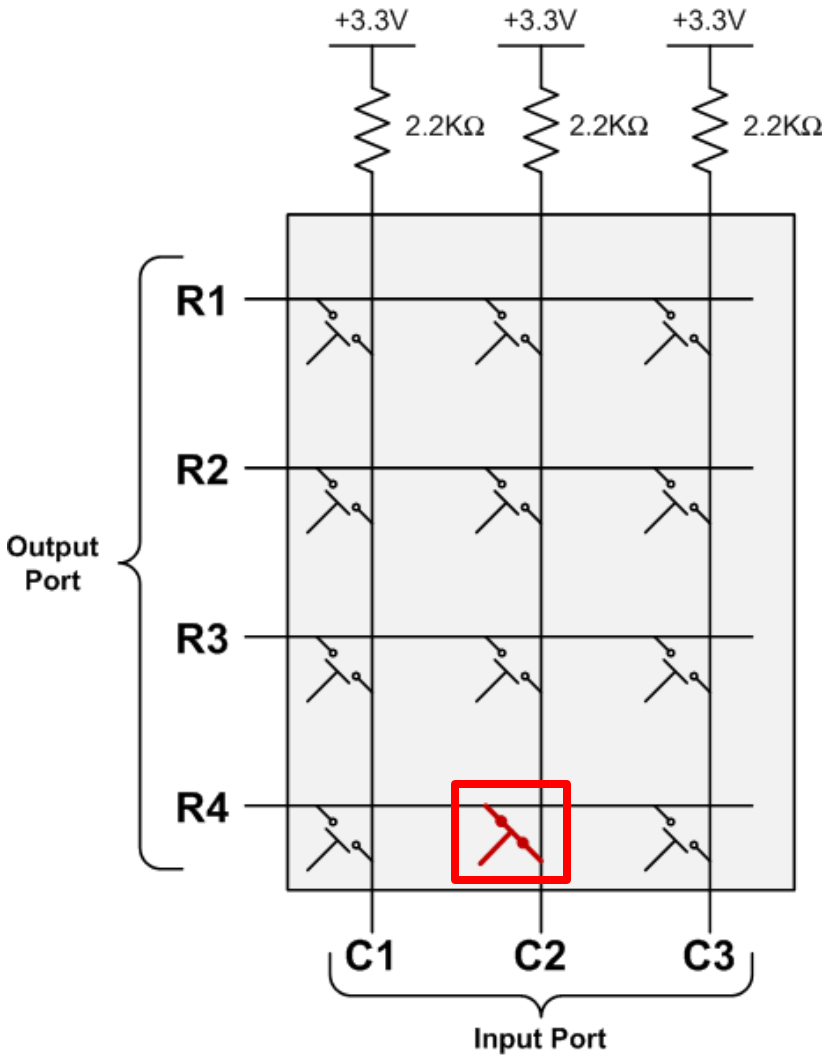
```
C1,C2,C3 = 111
```

⟹ No key pressed

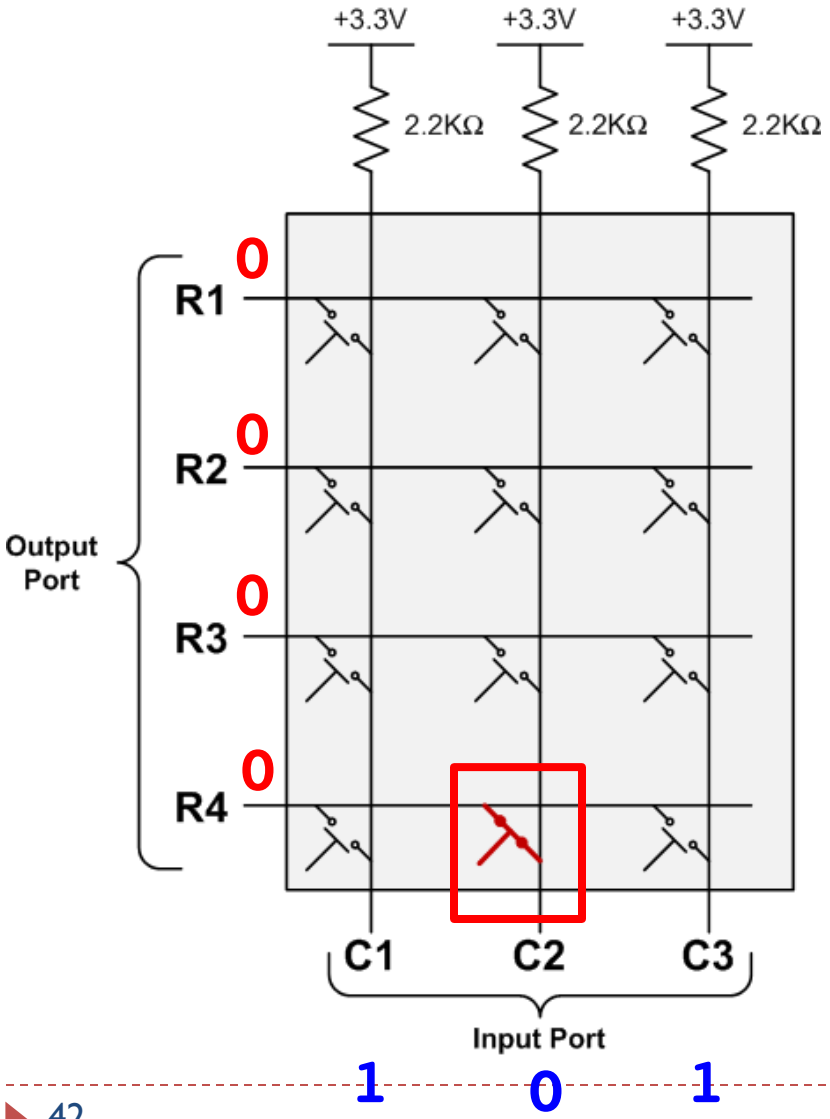
Keypad Scan



Key "0" is pressed



Keypad Scan



Step 1: Set Output

```
R1,R2,R3,R4 = 0000
```

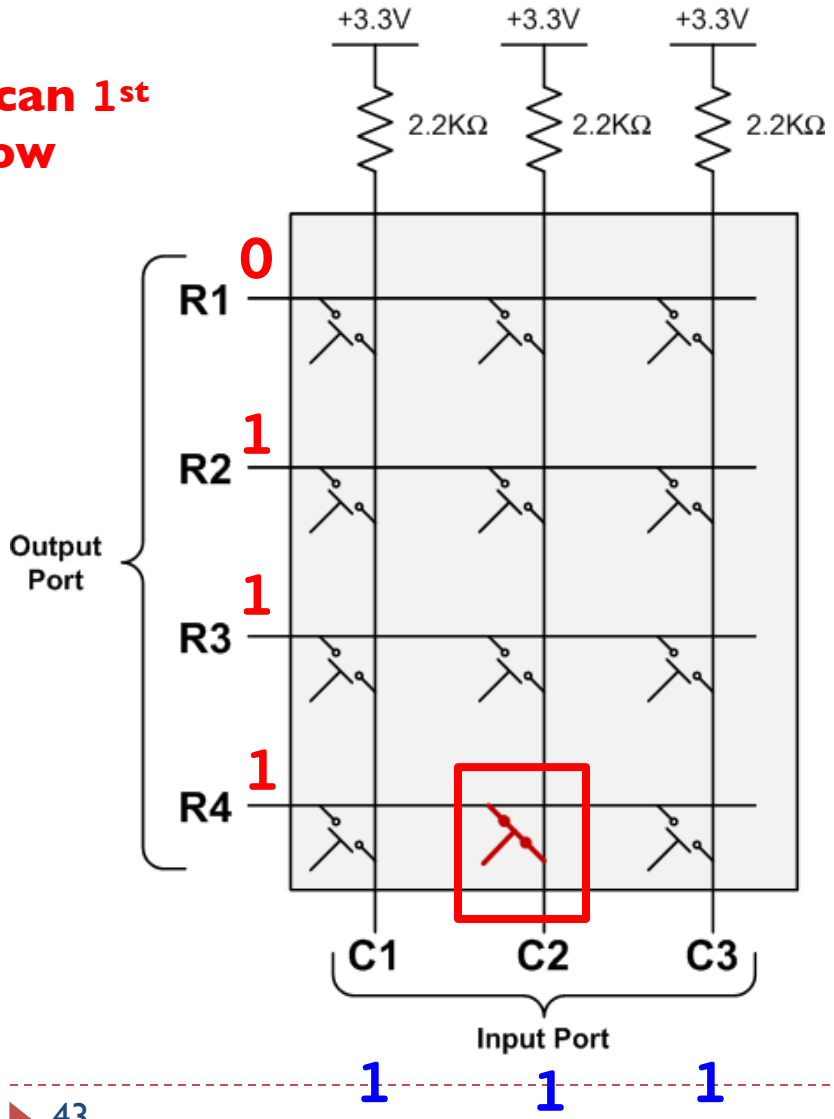
Step 2: Read Input

```
C1,C2,C3 = 101
```

⟹ Some key in 2nd column is pressed down

Keypad Scan

Scan 1st row



Step 1: Set Output

R1,R2,R3,R4 = 0000

Step 2: Read Input

C1,C2,C3 = 101

→

Step 3a: Scan 1st row

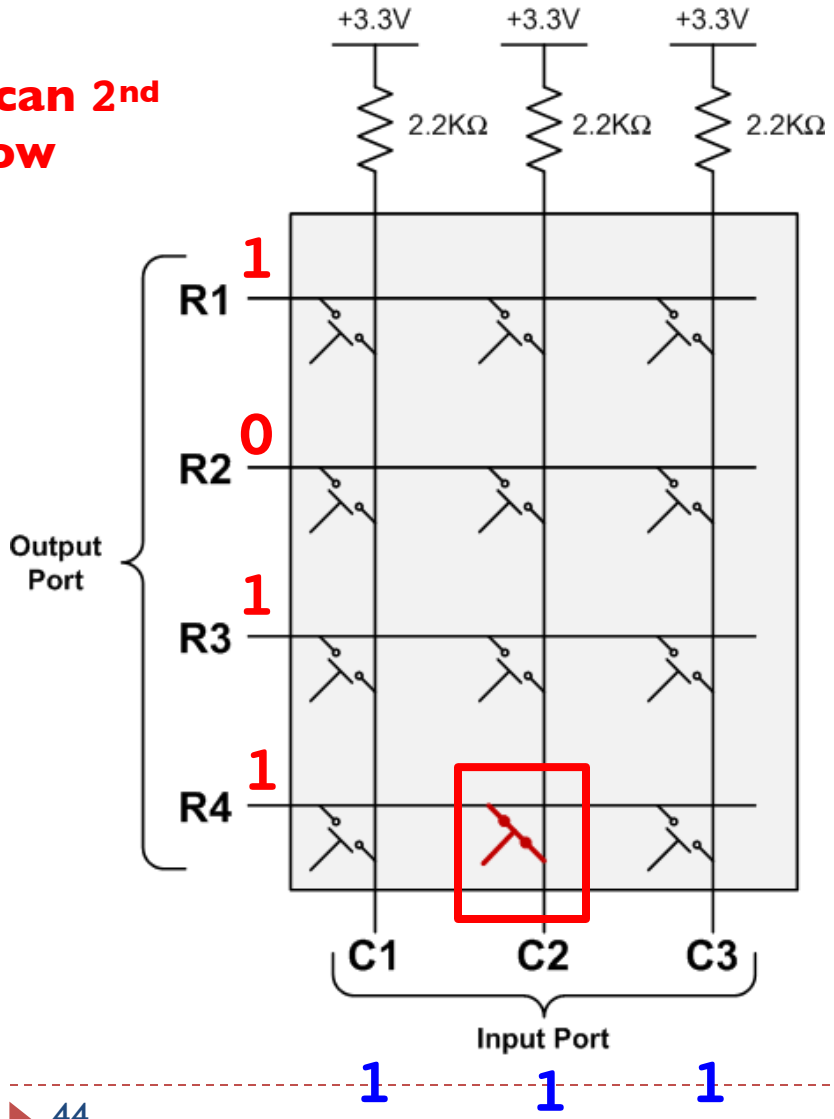
R1,R2,R3,R4 = 0111

C1,C2,C3 = 111

⟹ No key in 1st row is pressed down

Keypad Scan

Scan 2nd row



Step 1: Set Output

R1,R2,R3,R4 = 0000

Step 2: Read Input

C1,C2,C3 = 101

→

Step 3b: Scan 2nd row

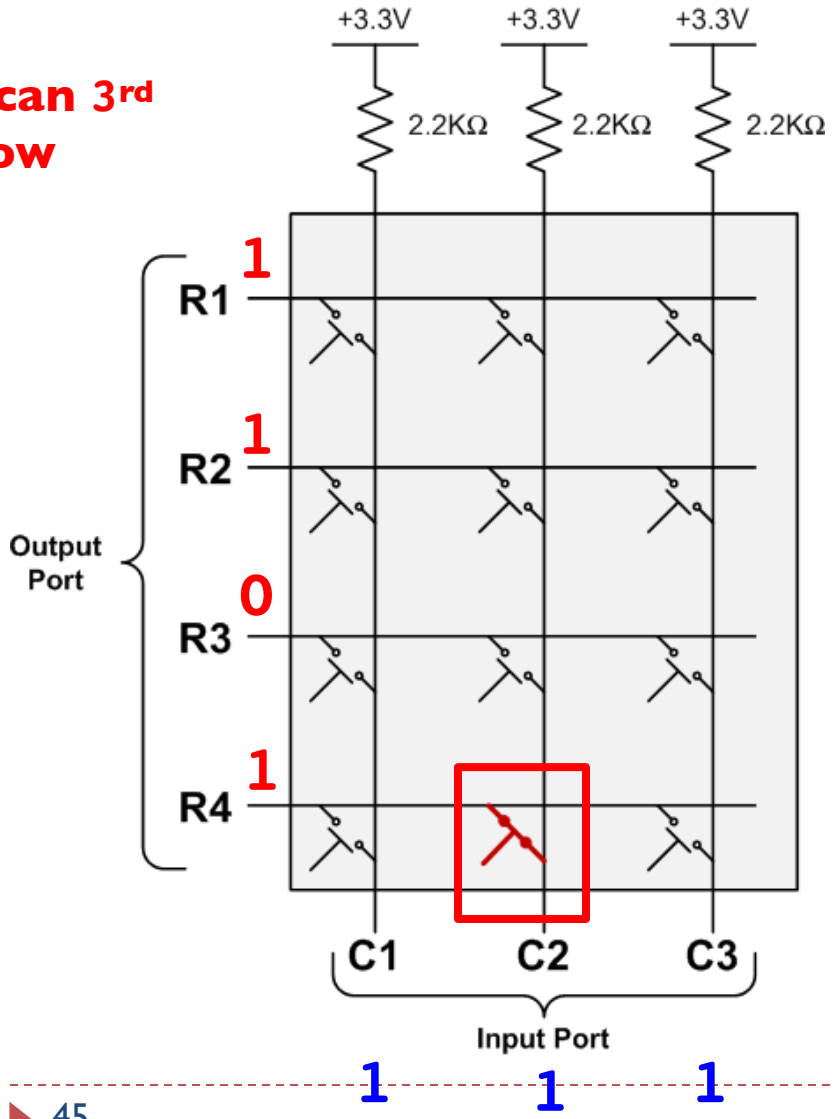
R1,R2,R3,R4 = 1011

C1,C2,C3 = 111

⟹ No key in 2nd row is pressed down

Keypad Scan

Scan 3rd row



Step 1: Set Output

R1,R2,R3,R4 = 0000

Step 2: Read Input

C1,C2,C3 = 101

→

Step 3c: Scan 3rd row

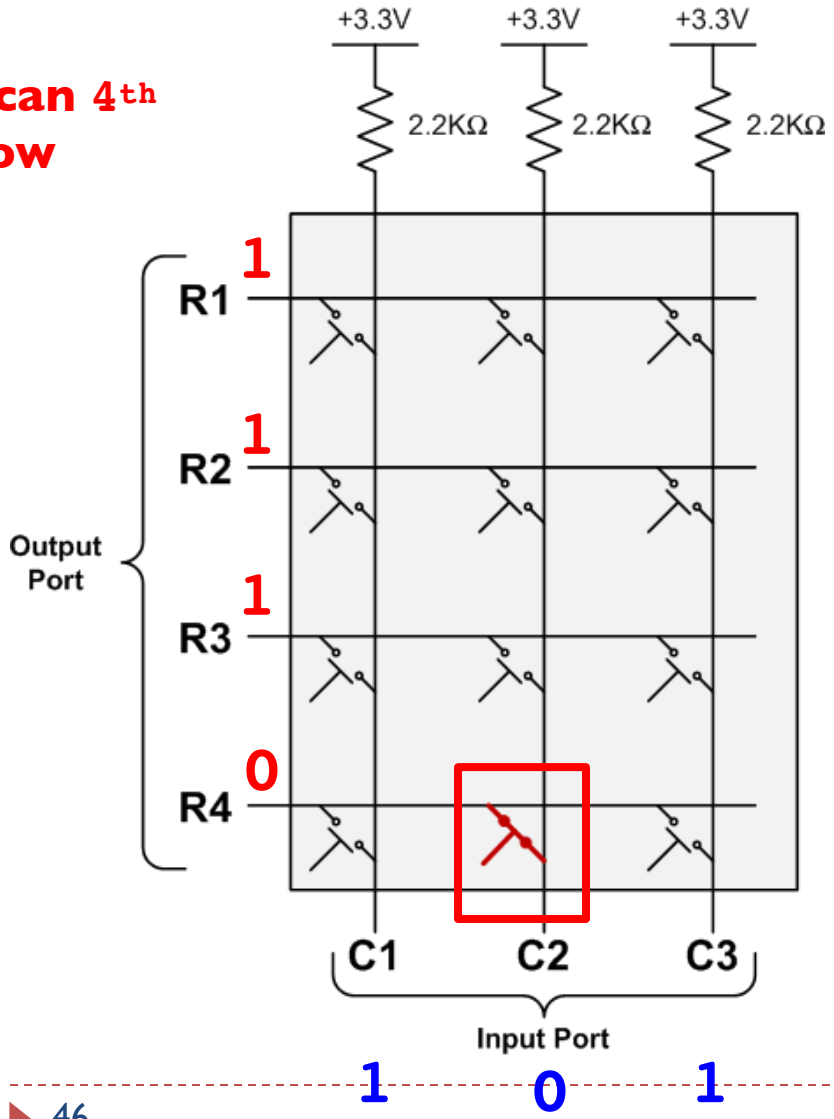
R1,R2,R3,R4 = 1101

C1,C2,C3 = 111

⟹ No key in 3rd row is pressed down

Keypad Scan

Scan 4th row



Step 1: Set Output

R1,R2,R3,R4 = 0000

Step 2: Read Input

C1,C2,C3 = 101

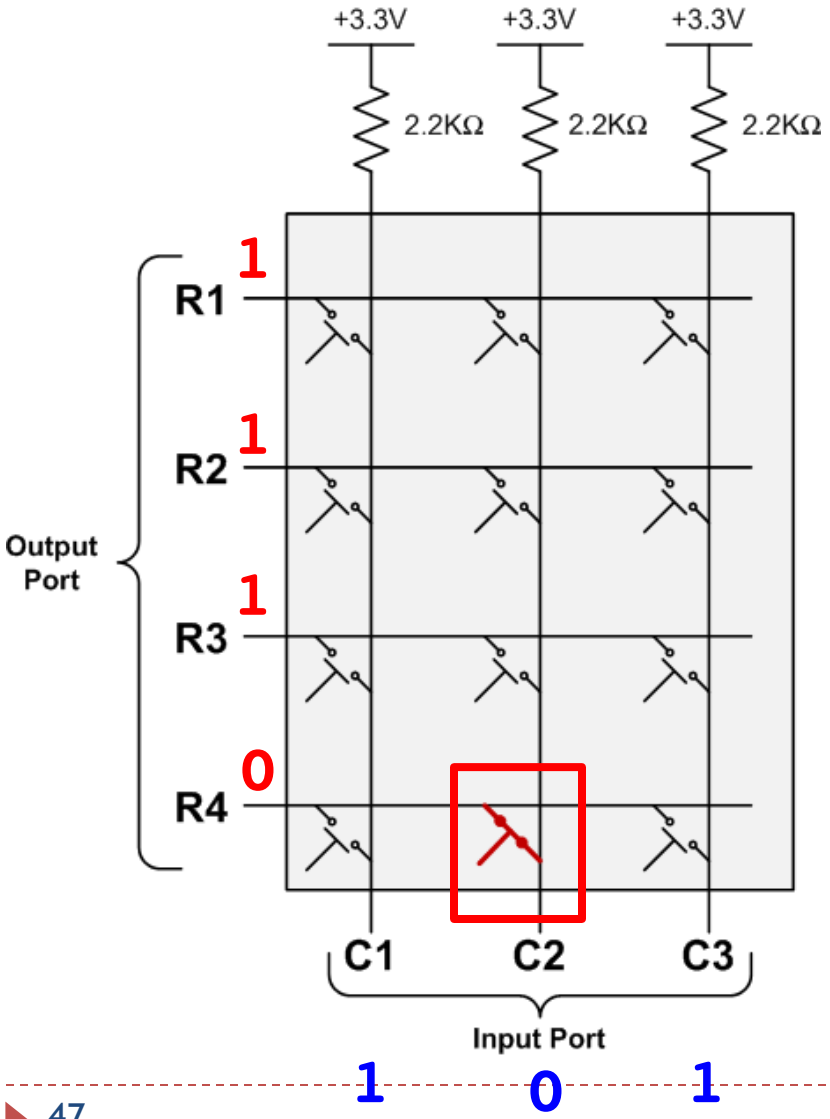
Step 3d: Scan 4th row

R1,R2,R3,R4 = 1110

C1,C2,C3 = 101

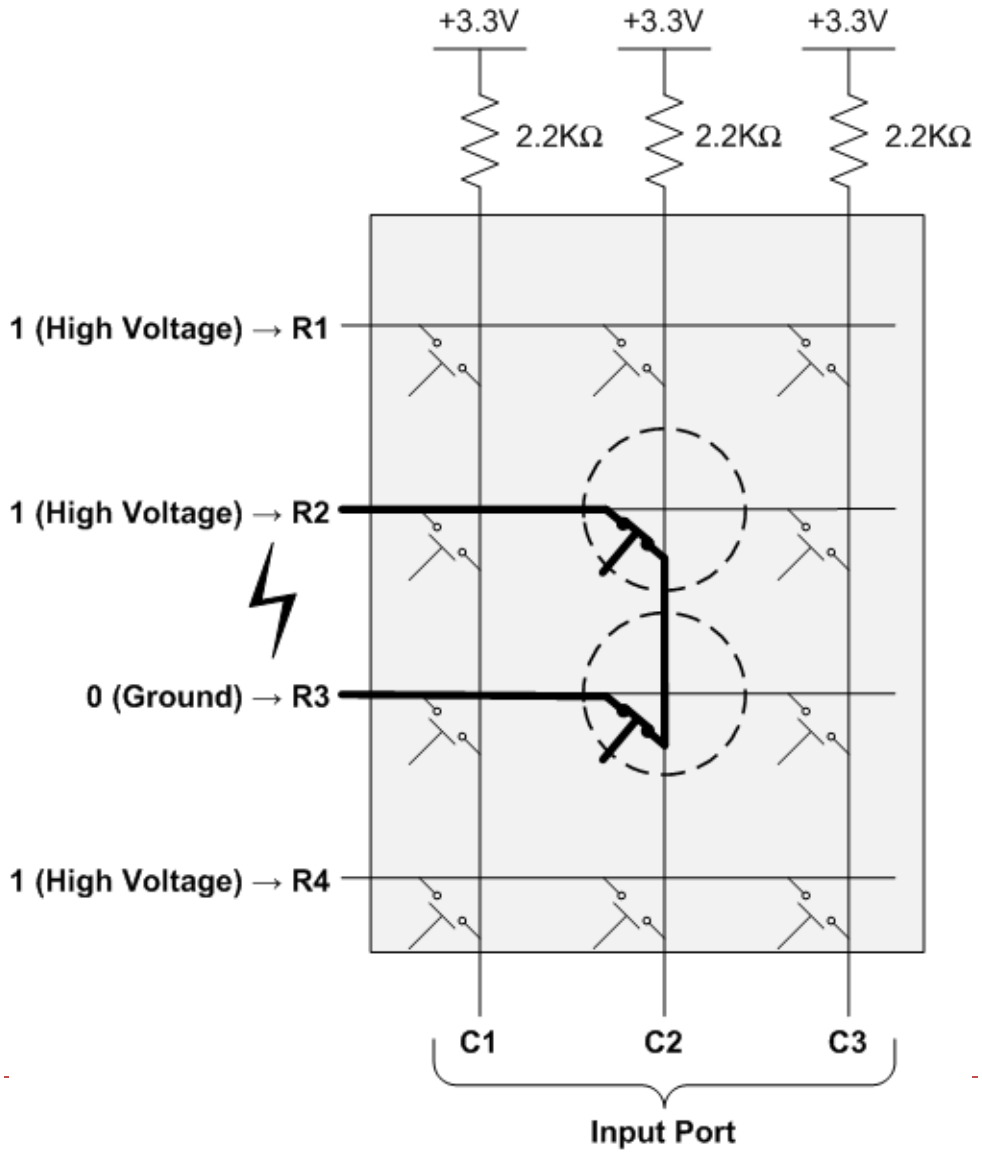
⟹ key in 4th row is pressed down

Keypad Scan



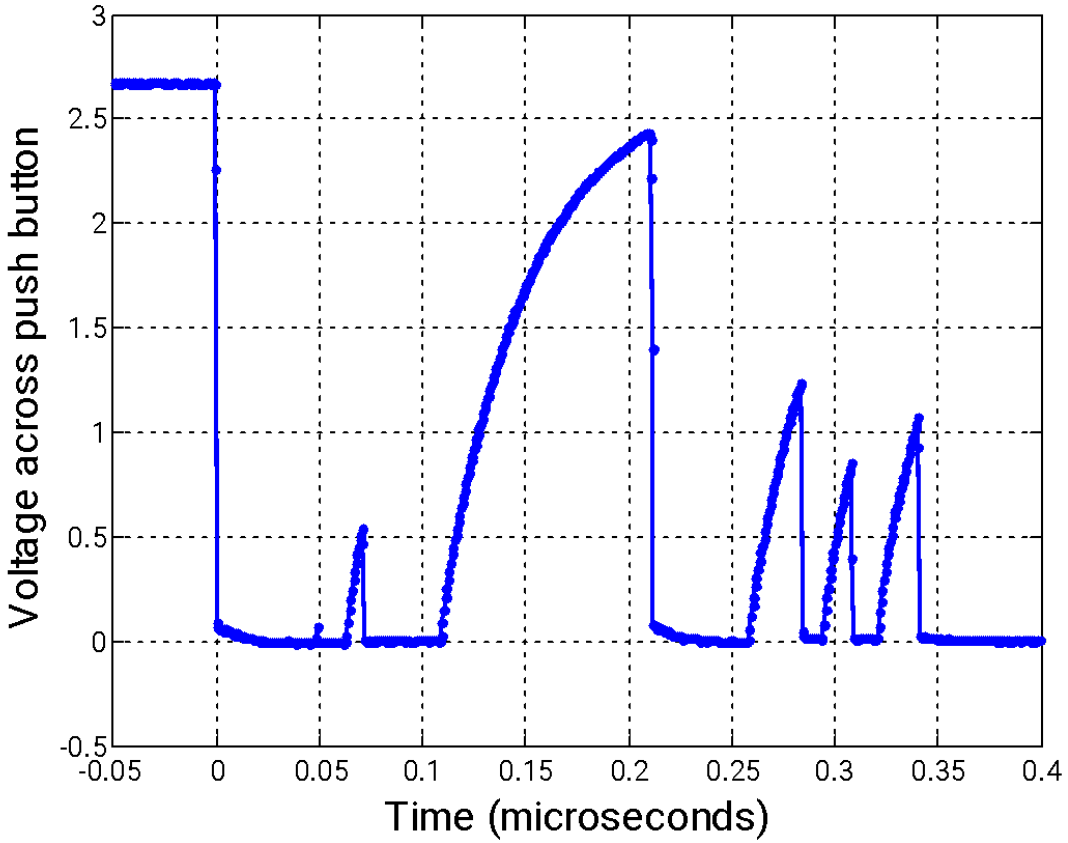
⟹ Key pressed is located at the second column and the fourth row.

Keypad Scan


+3.3V
+3.3V
+3.3V
2.2KΩ
2.2KΩ
2.2KΩ
1 (High Voltage) → R1
1 (High Voltage) → R2
0 (Ground) → R3
1 (High Voltage) → R4
C1
C2
C3
Input Port

I/O Debouncing

- Example signal when a button is pressed



END OF CLASS